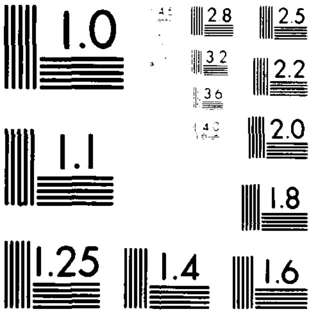
AD-A110 028   NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA   F/G 9/2
COMPUTER HARDWARE EXECUTIVE: CONCEPT AND HARDWARE DESIGN.(U)
SEP 81 D WILCOX
UNCLASSIFIED   NOSC/TR-721   NL

1 OF 1
AD A
110028

NOSC

END
DATE
FILMED
03-82
DTIC



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963 A

(12)

NOSC

NOSC TR 721

Technical Report 721

COMPUTER HARDWARE EXECUTIVE: CONCEPT AND HARDWARE DESIGN

Dwight Wilcox

5

September 1981

Produced for
NOSC IR/IED Program

Approved for public release; distribution unlimited

NAVAL OCEAN SYSTEMS CENTER
SAN DIEGO, CALIFORNIA 92152

NAVAL OCEAN SYSTEMS CENTER, SAN DIEGO, CA 92152

AN ACTIVITY OF THE NAVAL MATERIAL COMMAND

SL GUILLE, CAPT, USN
Commander

HL BLOOD
Technical Director

ADMINISTRATIVE INFORMATION

The work reported here was performed by members of the C^{3}I Systems Engineering Division under program element 62766N, project ZD78831A, and subproject ZF66212001, with funding provided by the NOSC IR/IED program.

Released by
S. Z. Mikhail, Head
C^{3}I Systems Engineering Division

Under authority of
V. J. Monteleon, Head
Command, Control, Communications and
Intelligence Systems Division

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NOSC Technical Report 721 (TR 721)	2. GOVT ACCESSION NO. AD-A110028	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) COMPUTER HARDWARE EXECUTIVE: CONCEPT AND HARDWARE DESIGN		5. TYPE OF REPORT & PERIOD COVERED 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dwight Wilcox		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Ocean Systems Center San Diego, CA 92152		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62766N, ZD78831A, ZF66212001
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE September 1981 13. NUMBER OF PAGES 51
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computers	SDEX/M
Hardware	AN/UYK-20
Executives	AN/AYK-14
Processing time	AN/UYK-44

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The potential of special-purpose hardware to improve execution speeds of computer executive functions is examined. Hardware designed to perform executive functions an order of magnitude faster than would be possible solely with software is described. SDEX/M, the Navy standard executive for the AN/UYK-20, AN/AYK-14, and the forthcoming AN/UYK-44 computers, is used as a working model and source of terminology.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

393159

EXECUTIVE SUMMARY

Large multiprocessing and distributed processing computer systems suffer from diminishing returns in system performance as additional processors are added. The slow execution speed of executive software is one of the principal causes of this phenomenon. The purpose of the executive software is to regulate the time when the various application programs gain access to the computer system resources. This task investigated the potential of special-purpose hardware to eliminate the execution-speed bottlenecks within executive software. A unit, named the Hardware Executive, was designed and fabricated.

The Navy standard SDEX/M executive was used as a model. Algorithms were developed for the executive functions of task creation, task dispatching, intratask coordination, real-time clock management, and event-to-task registration and translation.

The Hardware Executive employs an associative memory for high-speed manipulation of executive tables. It minimizes the number of context switches by performing the executive functions in hardware external to the host processor. Context switching, the transfer of the resources from one routine to another, is also minimized through executive algorithms that are insensitive to pre-emption. The Hardware Executive is interfaced to the host processor through its memory bus. This simple "memory-mapped" interface permits interfacing to a large number of existing computers without modification of their hardware or firmware. The interface also includes an interrupt for time-critical events when this feature is desired.

The Hardware Executive concept can benefit any commercial or military computer system where a large proportion of system capacity is lost because of software executive overhead.

CONTENTS

CONTENTS (Continued)

APPENDICES:

1. INTRODUCTION

An executive currently is defined as a program which regulates when each portion of the application software has access to the various resources of the computer. The application software informs the executive of its needs and available resources by calling executive service request (ESR) routines. Often several executive service requests share common subroutine components. The one or more subroutine components of an executive service request are called executive functions.

A number of studies [1--6] have been made in search of methods of significantly reducing the time spent in processing executive functions. Most of these studies have concentrated on using microprogramming to replace software bottlenecks. While the results vary because of differing architectures and algorithms [1,7], it appears that microprogrammed executive functions execute approximately three times faster than their respective assembly language implementations. The chief source of speed improvement is the elimination of nearly all instruction fetch cycles. Executive functions make heavy use of data structures stored in memory. Consequently, the speed of the data access cycle is the major limiting factor preventing still further speed improvement through microprogramming.

The purpose of this work is to examine the potential of special-purpose hardware to improve the execution speed of the executive functions. Hardware designed to perform executive functions an order of magnitude faster than would be possible solely with software is presented.

---

[1] Brown, George E., et al, Operating System Enhancement through Firmware, 10th Annual Workshop on Microprogramming (Micro-10), 1977, p 119-133.

[2] Burkhardt, Walter H., and Helmut E. Maier, Micos: A Microprogrammed Hierarchical Operating System Nucleus and Its Performance Comparison, 11th Annual Microprogramming Workshop (Micro-11), 1978, p 33.

[3] Burkhardt, Walter H., and Ronald C. Randel, Design of Operating Systems with Micro-Programmed Implementation, Pittsburgh University, Sept 1973.

[4] Chattergy, R., Microprogrammed Implementation of a Scheduler, 9th Annual Workshop on Microprogramming (Micro-9), 1976, p 15-19.

[5] Fogarty, J.R., et al, Hardware Command and Control System Study: Final Report, Hughes Aircraft Co, Ground Systems Group, Fullerton CA, 18 Feb 1974, Navy contract N00123-73-C-2130 CDRL A002 submitted to Naval Electronics Laboratory Center.

[6] Smith, William B., et al, SYMBOL - A large experimental system exploring major hardware replacement of software, SPRINT Joint Computer Conference, 1971, p 601-616.

[7] Dearnley, P.A., Application level microcode to speed data base management, Computer Journal, British Computer Society, v 22, no 3, Aug 1979, p 200-202.

SDEX/M [8], the Navy standard executive for the AN/UYK-20, AN/AYK-14, and forthcoming AN/UYK-44 general-purpose computers, is used as a working model and source of terminology. Instead of replacing the SDEX/M software entirely with hardware, attention is centered on extracting those executive functions most frequently invoked, for which the greatest execution speed advantage is expected. This paper presents a design to improve the performance of task creation, task dispatch, signal and wait, event registration, cause event, and real-time clock management functions. While not a part of SDEX/M, the Hardware Executive can also improve the speed of buffer pool management in a virtual memory system.

2. MOTIVATION FOR HIGH-SPEED EXECUTIVE FUNCTIONS

The throughput of a multiprocessing system [5,9,10] increases less than linearly with an increase in the number of processors. Each processor is burdened not only with the coordination of its own activity but also with at least some portion of the activity of the processors to which it is connected. Increasing the number of processors increases the overhead in each processor. Eventually, a point is reached where adding a processor has a negligible effect or even decreases the usable throughput of the system. This is because the increased overhead in all the processors totals more than the capacity of the added processor.

An admittedly oversimplified example serves to illustrate the effect. One of the major manifestations of executive overhead is table searching (figure 1). It is encountered in dispatch checking, real-time clock management, resource management, and event binding in a hierarchical event system. Consider a multiprocessor system employing a common (shared) table. A configuration consisting of a single stand-alone processor spends, say, 10 percent of the time serially searching the table. Now consider adding a second processor to improve system capacity. When the second processor is added, each processor must examine the same table, which is now roughly twice as large. As a consequence, the time each processor spends searching the table is roughly doubled. Actually, the time is slightly less than doubled since, with more time spent searching the table, there is less time available for other processing. With slightly less processing, the table is also slightly smaller. The actual portion of time spent searching the table is computed by normalization.

$$\frac{2 \text{ X } 0.10}{2 \text{ X } 0.10 + (1 - 0.10)} = 18 \text{ percent.}$$

[8] Naval Electronic Systems Command, Computer Program Performance Specification for Standard Executive for use with AN/UYK-20 and AN/AYK-14 Computers, SDEX/M; NAVSEA 0967-LP-598-2710, change 1, Dec 1980.

[9] Chen, Tien Chi, Parallelism, Pipelining, and Computer Efficiency, Computer Design, Jan 1971, p 69-74.

[10] Wulfinghoff, Donald R., Code Activated Switching: A Solution to Multiprocessing Problems, Computer Design, Apr 1971, p 67-71.

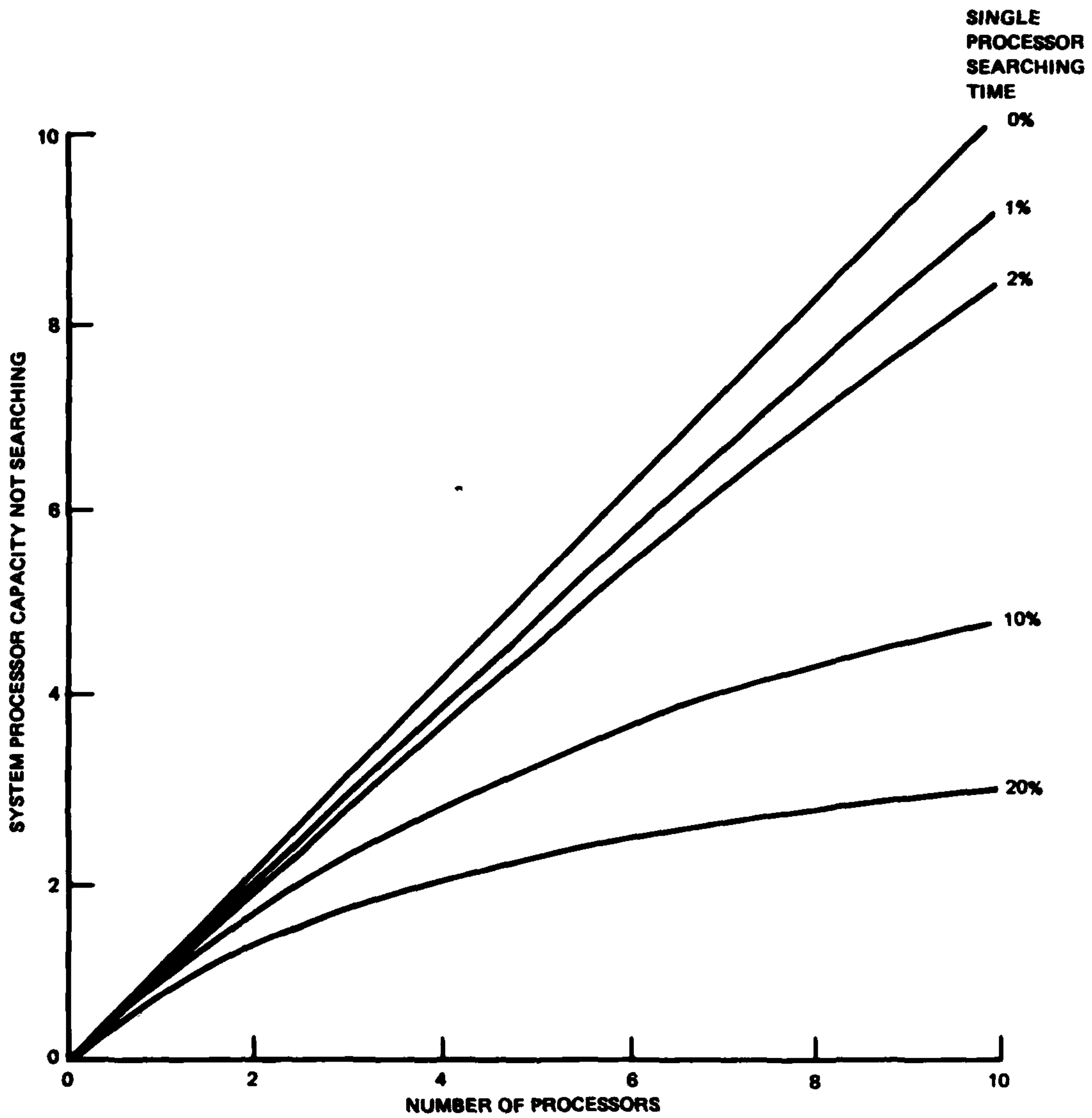


Figure 1. Multiprocessing system where all processors search a common table.

The time each processor spends searching the table continues to increase as more processors are added to the system.

NUMBER OF PROCESSORS IN SYSTEM	PERCENTAGE OF TIME SEARCHING	PERCENTAGE OF TIME NOT SEARCHING	SYSTEM CAPACITY NOT SEARCHING
1	10	90	0.90
2	18	82	1.64
3	25	75	2.25
4	31	69	2.76
5	36	64	3.20

Since table searching is an important component determining the speed of executive processing, executive software often exhibits similar characteristics. A good example is a common task table in a multiprocessor in which the highest-priority task ready for execution is dispatched on the first available processor. The dispatch check performed by each processor must examine all the tasks in the system. The larger the system, the more tasks each processor must examine.

Message processing in a distributed processing system also suffers from diminishing returns as processors are added. One example is SEAMOD [11], a proposed Navy distributed command and control system architecture. The processors of the system are interconnected through a high-speed message bus. The table below [12] shows the processor capacity required for message processing on various configurations.

CONFIGURATION		PROCESSORS REQUIRED			
System	Bus	Executive	Application	Message Overhead	Percent Increase
SEAMOD LOMIX (FFGX)	Broadcast	SDEX-20	37	18.5	50.0 %
		SDEX-M		11.2	30.3 %
	Point to Point	SDEX-20		20.5	55.4 %
		SDEX-M		12.1	32.7 %
SEAMOD HIMIX (DD 963)	Broadcast	SDEX-20	48	25.0	52.1 %
		SDEX-M		15.1	31.5 %
	Point to Point	SDEX-20		27.7	57.7 %
		SDEX-M		16.3	34.0 %

The only overhead included in the figures is from the executive functions of I/O chain initialization, interrupt-to-event binding, and dispatch of the buffer allocator and application tasks. Execution of the buffer allocator and I/O handlers is not included. As can be seen from the table, from 30.3 to 57.7 percent more processors are required, depending on the configuration, simply to perform the executive functions associated with message traffic of the distributed system.

[11] Mundell, J.L., et al, Conceptual Design of a Distributed Combat Direction System for a Modular Frigate (SEAMOD FFGX), prepared by System Exploration, Inc, under Navy Contract N00123-76-C-0787, Feb 1978, for Naval Ocean System Center, NOSC Technical Note 356.

[12] Sperry Univac, Defense Systems Division, Navy Standard Computer Adaptation Study, v 1: Application Requirements, Navy Contract N66001-78-C-0258, July 1979, Final Report, prepared for NOSC.

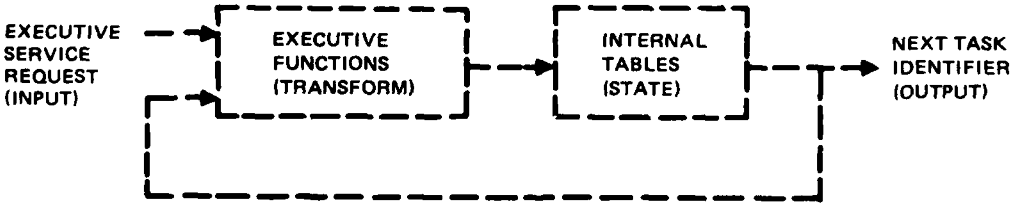
Another example of the problem is illustrated by an experiment performed by the developers of the Shipboard Integrated Processing and Display System (SHINPADS) [13] for the Canadian Navy. They wished to demonstrate 2 megabits of data traffic on a triaxial cable bus interconnecting 10 AN/UYK-20 emulators. Each computer generated and received one-tenth of the 2-megahertz traffic. Using 200-bit messages, this meant that each computer generated and received one message every 1000 microseconds. It was found that the 1000 microseconds was consumed almost entirely by the executive software. Not only does such a system fail to apply the power of 10 computers to the performance of application work; it fails to do almost any useful work.

The problem is that the execution speed of the executive functions is too slow in a conventional processor. The executive functions considered in the SEAMOD example, when executed on an AN/UYK-20 processor, require 1120 microseconds per message using SDEX/20 or 660 microseconds per message using SDEX/M. The motivation for high-speed execution of executive functions in hardware is the reduction of overhead in each processor of a multiprocessor or distributed processor system to extend the point of diminishing return to a larger number of computers.

3. NATURE OF THE EXECUTIVE IMPLEMENTATION PROBLEM

A number of presently implemented software executives were examined to determine where they spend their time. Three basic areas were identified which consume a large proportion of time: linked-list manipulation, context switching, and error checking.

Table and linked-list manipulation represent the actual service provided by the executive. Such manipulation is fundamentally a software implementation of what hardware engineers call a state machine. In the state machine model, the executive service request (ESR) is the input, the identity of the next running task is the output, and the content of the various internal tables is the machine state. Executive functions are the algorithms used to transform the present state into the next state.


EXECUTIVE SERVICE REQUEST (INPUT)
EXECUTIVE FUNCTIONS (TRANSFORM)
INTERNAL TABLES (STATE)
NEXT TASK IDENTIFIER (OUTPUT)

[13] Kuhns, Richard C., A Serial Data Bus System for Local Processing Networks, 18th IEEE Computer Society International Conference, Spring 1979, IEEE Catalog no 79CH1393-8C, p 266-271. (SHINPADS is also discussed in the Naval Engineering Journal, April and June 1979.)

One of the ways the Hardware Executive improves executive performance is by implementing this state machine problem in hardware specifically designed for the purpose. An associative memory is used to quickly locate the current state of the task or event referenced by the input executive service request. Additional hardware is provided to manipulate the state at the bit rather than word level.

Context switching is a form of overhead encountered when resources occupied by one routine, typically the processor and its registers, are transferred to another routine. One source of context switching is the transfer from the application software to the executive software and back again. Hardware implementation of the executive can reduce this form of context switching by providing separate resources for the executive which are independent of those used by the application software. Another form of context switching results from the specification of re-entrant executive functions. An executive function is made re-entrant to permit its pre-emption by an interrupt handler desiring utilization of the same function. Future experience may show that the existence of executive functions implemented in hardware that run an order of magnitude faster than their software counterparts eliminates much of the requirement for pre-emption. But even if this proves false, the Hardware Executive has been designed to allow pre-emption. This is accomplished by using the same principle employed in the "test and set" (Biased Fetch in the AN/UYK-20) instructions of many modern computers.

Error checking can be divided into software development debug aids and operational self-preservation when the system is overloaded. Code for the first type of error is often complex and not very structured. By definition, it is seldom executed after the system has been developed. For these reasons and because of the difficulty of implementing it in hardware, emphasis was placed on the second type of error.

4. SOFTWARE INTERFACE

The software programmer communicates with the Hardware Executive by using a technique commonly known as "memory mapping." Each executive function is assigned a dedicated address in the memory address space of the computer. The particular address accessed implies the function to be performed. The data read from or written to the executive function address is the operand of the executive function. The DISPATCH CHECK function, for example, gives the identity of the highest-priority task that should obtain control of the computer. It is performed by reading the contents at the dedicated DISPATCH CHECK function address.

The Hardware Executive maintains task and event tables internally. These are also mapped into the computer memory address space. They can be read or written like conventional memory. The user must not write into them indiscriminately, however, since their contents have an impact on the proper operation of the algorithms implementing the executive functions.

4.1 TASK MANAGEMENT

A task is created by specifying an entry for the task in the internal task table of the Hardware Executive. First, an empty location in the task table must be obtained by reading from the dedicated RESERVE TASK executive function address. The RESERVE TASK function returns to the host processor a positive index into the task table of an empty task table location. If no empty task table locations are available, the function returns to the host processor a negative number instead. This index is then used to directly load the task parameters into the task table entry.

A task known to the executive is in one of five possible states at any given time.

Running: The task is executing on the host processor.

Ready: The task can execute on the host processor.

Waiting: The task cannot execute on the host processor until cleared to do so by a flag called a semaphore.

Suspended-Ready: The task is explicitly prevented from executing on the host processor.

Suspended-Waiting: The task is explicitly prevented from executing on the host processor, but if it were not explicitly prevented, it would still be unable to execute until cleared to do so by a flag called a semaphore.

The RESERVE TASK function initializes the task to the Suspended-Ready task state to prevent accidental dispatch until the task table entry is completely specified.

The task management executive functions control the movement of a task from one state to another (figure 2).

The DISPATCH CHECK function selects, from among the tasks in the Ready and Running task states, the task with the highest priority. The priority is a positive number, supplied by the user, which indicates the relative importance of executing the associated task before other tasks when a choice is possible. A low numeric value represents a high priority. If two or more tasks with the same priority are in the Ready or Running task states, they are executed on a first-come-first-served basis. The DISPATCH CHECK function returns to the host processor the index in the task table of the task placed in the Running state. The host processor then uses this index to perform the task-to-task context switch. If no task can be placed in the Running state, the DISPATCH CHECK function returns a negative number to the host processor. This directs the host processor to loop in an idle state, awaiting activity.

The WAIT function is used to move a task in the Running task state into the Waiting task state if the resource it desires is not available. For example, the user may desire that only one task at a time have access to a

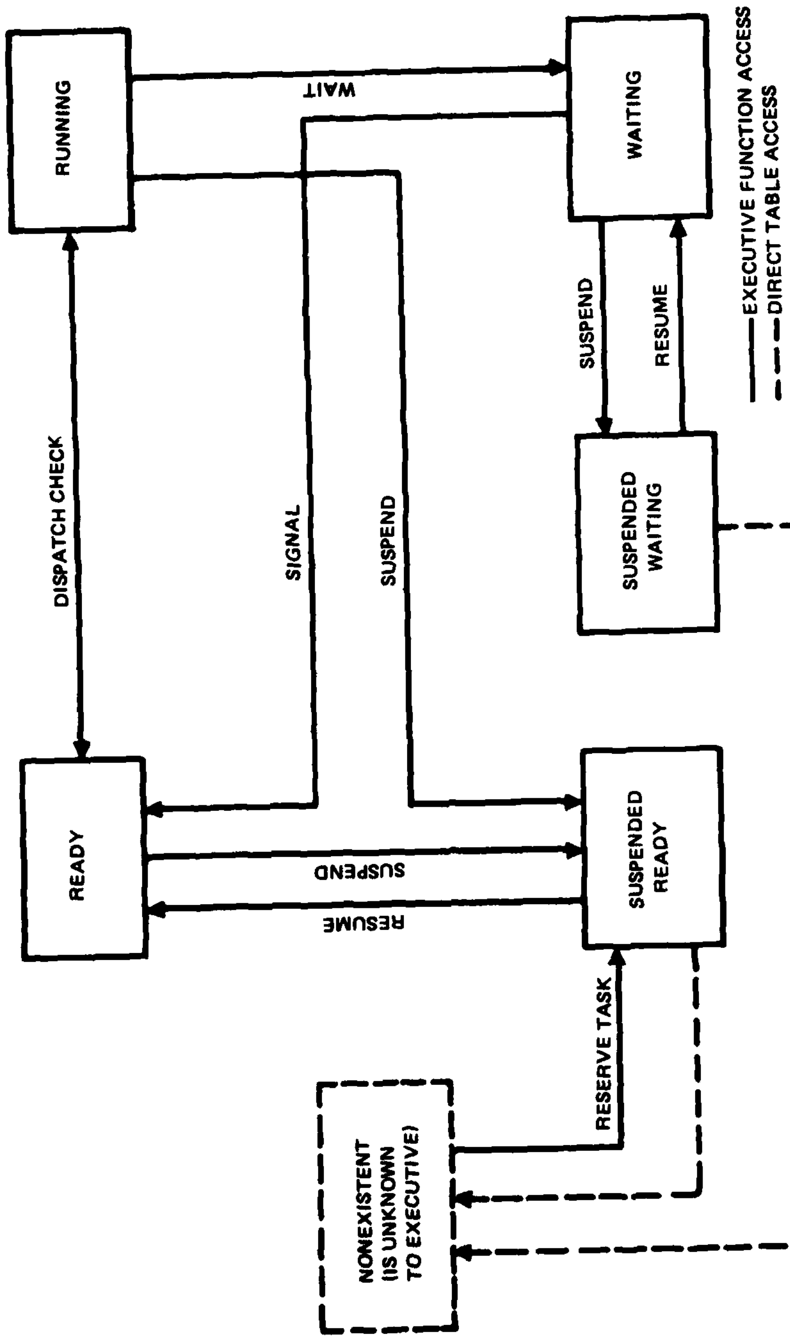


Figure 2. Hardware executive task state diagram.

printer. All tasks invoke the WAIT function to determine whether the resource, in this case the printer, is available before an attempt is made to access it. Availability of a resource is indicated by a flag called a semaphore. The semaphore for each resource is given a unique Semaphore Identifier by the user. The WAIT function first checks the Semaphore field of all the nonsuspended tasks in the task table to determine whether any other task is using the semaphore specified by the operand of the WAIT function. If none is found, the resource is available and the task invoking the WAIT function can remain in the Running task state. Otherwise, the task is moved from the Running task state to the Waiting task state. In either case, the Semaphore Identifier specified by the operand of the WAIT function is stored in the task table entry of the task that invoked the WAIT function. If the resource is available, this claims the resource for the task that invoked the WAIT function. If the resource is not available, it puts the task on the waiting list. The WAIT function should be followed by a DISPATCH CHECK function since the task currently in the Running task state may now be in the Waiting task state.

The SIGNAL function is used to announce that a resource is now available. It first searches for the highest priority nonsuspended task waiting on the semaphore specified by the SIGNAL function operand. Once found, the task is moved from the Waiting task state to the Ready task state and its Semaphore field in the task table is cleared. The SIGNAL function should be followed by a DISPATCH CHECK function since the task just placed in the Ready task state may have a higher priority than the task currently in the Running task state.

The SUSPEND function is used to explicitly exclude a specified task from consideration by the DISPATCH CHECK function. If the task is in the Ready or Running task state, it is moved to the Suspended-Ready task state. If the task is in the Waiting task state, it is moved to the Suspended-Waiting task state. The SUSPEND function should be followed by a DISPATCH CHECK function to obtain the identity of the next task in the Running task state.

The RESUME function is the converse of the SUSPEND function. If the task is in the Suspended-Ready task state, it is moved to the Ready task state. If the task is in the Suspended-Waiting task state, it is moved to the Waiting task state. Since the resource required by a task in the Suspended-Waiting task state may be available by the time the RESUME function is invoked, the semaphore test is repeated. The RESUME function should be followed by the DISPATCH CHECK function since a task now in the Ready task state may have a higher priority than the task currently in the Running task state.

To destroy a task, the user must first execute the SUSPEND function to properly exclude the task from further dispatch. Once the task is in the Suspended-Ready or Suspended-Waiting task state, it is ignored by all executive functions except the RESUME function. In this state, the task table entry for the task can be cleared directly.

4.2 EVENT MANAGEMENT

The Hardware Executive distinguishes among normal events and time events (figure 3). A normal event is an event triggered explicitly by the user with the CAUSE NORMAL EVENT executive function. A time event is an event triggered by a preset time interval.

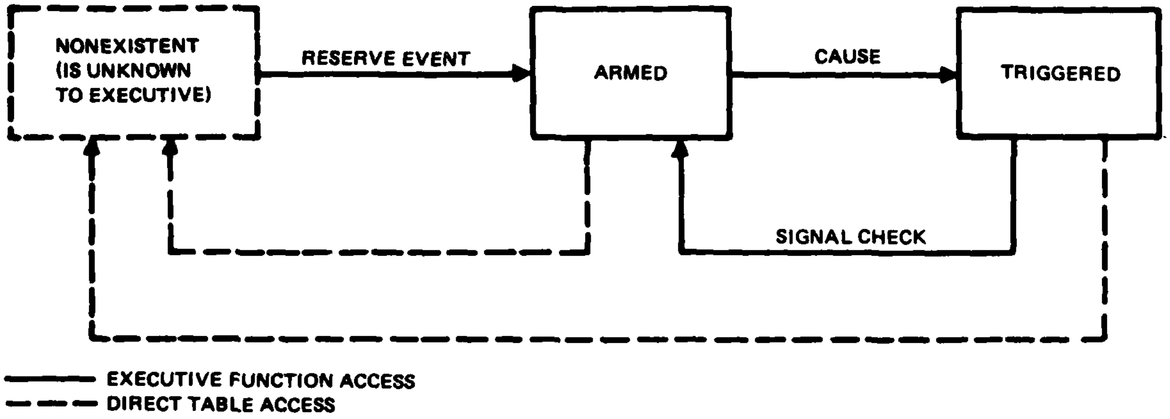


Figure 3. Hardware executive event state diagram.

Normal and time event registrations are maintained in separate tables within the Hardware Executive. An event is registered by first obtaining an empty event table location by means of either the RESERVE NORMAL EVENT or the RESERVE TIME EVENT functions. These functions return a positive index to the host processor by which the user can load the table entry. Return of a negative number indicates that no empty table locations are available.

An event known to the executive has one of two possible states at any given time.

Armed: The executive is ready should the event occur.

Triggered: The event has been recognized but the semaphore associated with the event is still not signalled.

The CAUSE NORMAL EVENT function matches the event specified by the function operand with a qualifying event registration in the normal event table. The event registration selected is then moved from the Armed event state to the Triggered event state. The CAUSE NORMAL EVENT function should be followed immediately by a NORMAL EVENT SIGNAL CHECK function. The NORMAL EVENT SIGNAL CHECK function signals the semaphore associated with the highest priority normal event registration in the Triggered event state. A negative number is returned to the host processor when no event registrations remain in the triggered state. The NORMAL EVENT SIGNAL CHECK function, in turn, should be followed by a DISPATCH CHECK function since the task just signalled may have a higher priority than the task currently in the Running task state.

Time events are triggered automatically by a Real-Time Clock within the Hardware Executive. The TIME EVENT SIGNAL CHECK function performs the same function for time events as the NORMAL EVENT SIGNAL CHECK function performs for normal events. It should be invoked periodically by the user. Since several tasks may be signalled at the same time, the function should be repeated until it returns a negative result. Finally, the DISPATCH CHECK function should be performed to select the highest-priority task for execution.

5. HARDWARE DESIGN

This section first presents the external functional interface of the Hardware Executive, followed by a detailed description of its internal logic design. The executive functions and their implementation algorithms are then presented. Finally, the microprogram algorithms used to implement the various executive functions are presented.

5.1 HOST PROCESSOR INTERFACE

The Hardware Executive interfaces to a host processor through the memory interface of the host processor. This is a common technique known as "memory mapping." The programmer selects the executive function to be performed by specifying the memory address of the function. The operand associated with the executive function is passed as data read to or written from that address. For example, a task DISPATCH CHECK function is executed by reading the task identifier from the dedicated DISPATCH CHECK function address. A SIGNAL semaphore function is executed by writing the semaphore identifier into the dedicated SIGNAL function address.

The Hardware Executive maintains internally a task table, a normal event table, and a time event table. These tables are also mapped into the host processor address space. This enables the host processor to initialize the task or event state after the Hardware Executive has provided an empty table entry. A task is created when the host processor reads a new task identifier from the RESERVE TASK function address. The task identifier is equivalent to an index into the task table address space. The host processor can then use this index to directly initialize the task priority, state, and semaphore fields. Normal and time events are created in a similar manner by using the RESERVE NORMAL EVENT and RESERVE TIME EVENT functions.

The Hardware Executive utilizes associative memory to store the task and event tables. An associative memory, often called a content-addressable memory, is a memory which, when given the content of one or more of its locations, returns the address or addresses of the locations with those contents. The associative memory thus implements a table search in hardware. The associative memory locations are called elements to avoid confusion with other uses of the word location. The associative memory of the Hardware Executive is designed to permit searches on any bit position or set of bit positions within its elements. For example, one bit position of the task table locations indicates whether the table location is empty or contains a task table entry supplied by the user. The RESERVE TASK function searches on this bit position to find an empty table location.

In addition to the interface through the memory bus of the host processor, the Hardware Executive also connects to the host processor through an interrupt it generates for time-critical events. This interface is only necessary if time-critical event processing by the Hardware Executive is desired.

5.1.1 AN/UYK-20 Interface

The AN/UYK-20 computer was not designed for expansion beyond 64k words of memory. Consequently, an off-the-shelf AN/UYK-20 must be slightly modified internally before the Hardware Executive can be connected.

The breadboard built to demonstrate the Hardware Executive concept was interfaced to an AN/UYK-20 (without the DMA option) by using the DMA connector and card slot (figure 4). A new printed circuit board was designed for the DMA card slot to act as an interface between the internal AN/UYK-20 memory bus and the Hardware Executive connected through the DMA connector. The DMA connector provides a convenient means of getting through the box without mechanical modifications. The interface card compares the address of all memory accesses with the dedicated address space of the Hardware Executive. The dedicated address space is programmed on the card through straps. When a match occurs, the AN/UYK-20 READ INITIATE signal, which is daisy-chained through the interface card, is inhibited, thus preventing the AN/UYK-20 core memory from receiving a memory initiate request. This technique effectively disconnects the core memory for those addresses dedicated to the Hardware Executive.

5.1.2 AN/AYK-14 Interface

The AN/AYK-14 CPU provides the option of interfacing to either the virtual or the physical address space. Virtual addresses from the CPU are paged into physical addresses by the Memory Control Module (MCM). There are two data busses, called "M" and "X", connecting the CPU to the MCM. If the Hardware Executive is interfaced within the virtual address space, then it must be made dual-ported so that memory accesses from either bus will be processed. The physical address space can also be configured into two busses to permit interleaving. Here, the need for a dual-ported interface is not as critical since one could map the Hardware Executive address space into only even words, leaving the odd words unused. One advantage of the AN/AYK-14 is that its physical address space, 512k words, is large enough that leaving some unused memory locations does not represent much of a percentage loss in the entire address space.

Unlike the AN/UYK-20, the AN/AYK-14 has an external expansion connector with all the necessary signals present.

5.2 INTERNAL CONSTRUCTION

The design of the Hardware Executive is functionally divided into four parts: Associative Memory Logic, Memory Selection Logic, Register Logic, and Microsequence Control Logic.

5.2.1 Associative Memory Logic

The Associative Memory Logic is used to implement the storage and search of executive tables. A block diagram of the Associative Memory Logic is shown in figure 5.

The heart of the Associative Memory Logic is a conventional random access memory (RAM). The RAM is constructed to provide independent read/write control of each bit position of the addressed word. This is accomplished by

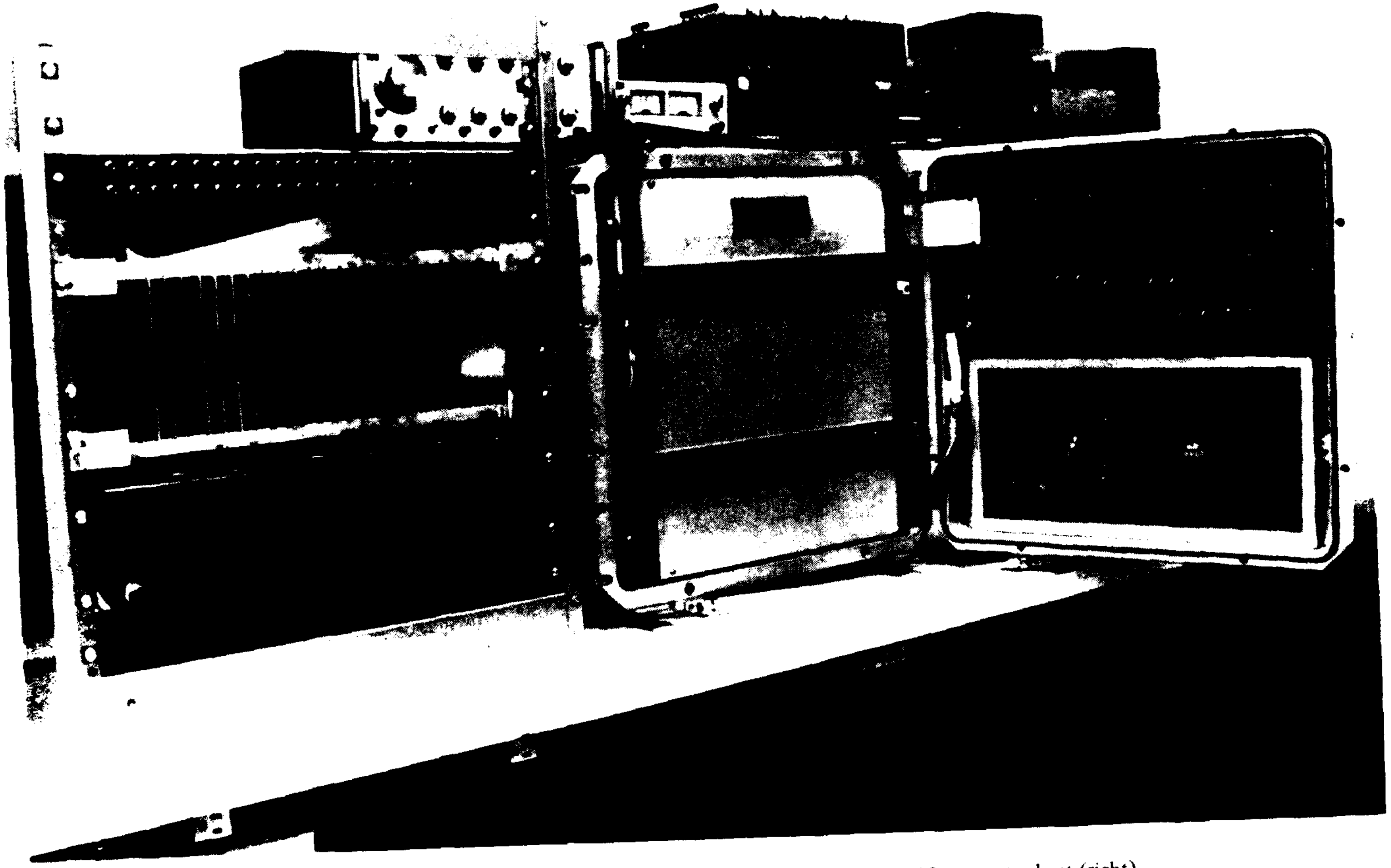


Figure 4. Hardware Executive breadboard (left) connected to AN/UYK-20 computer host (right).

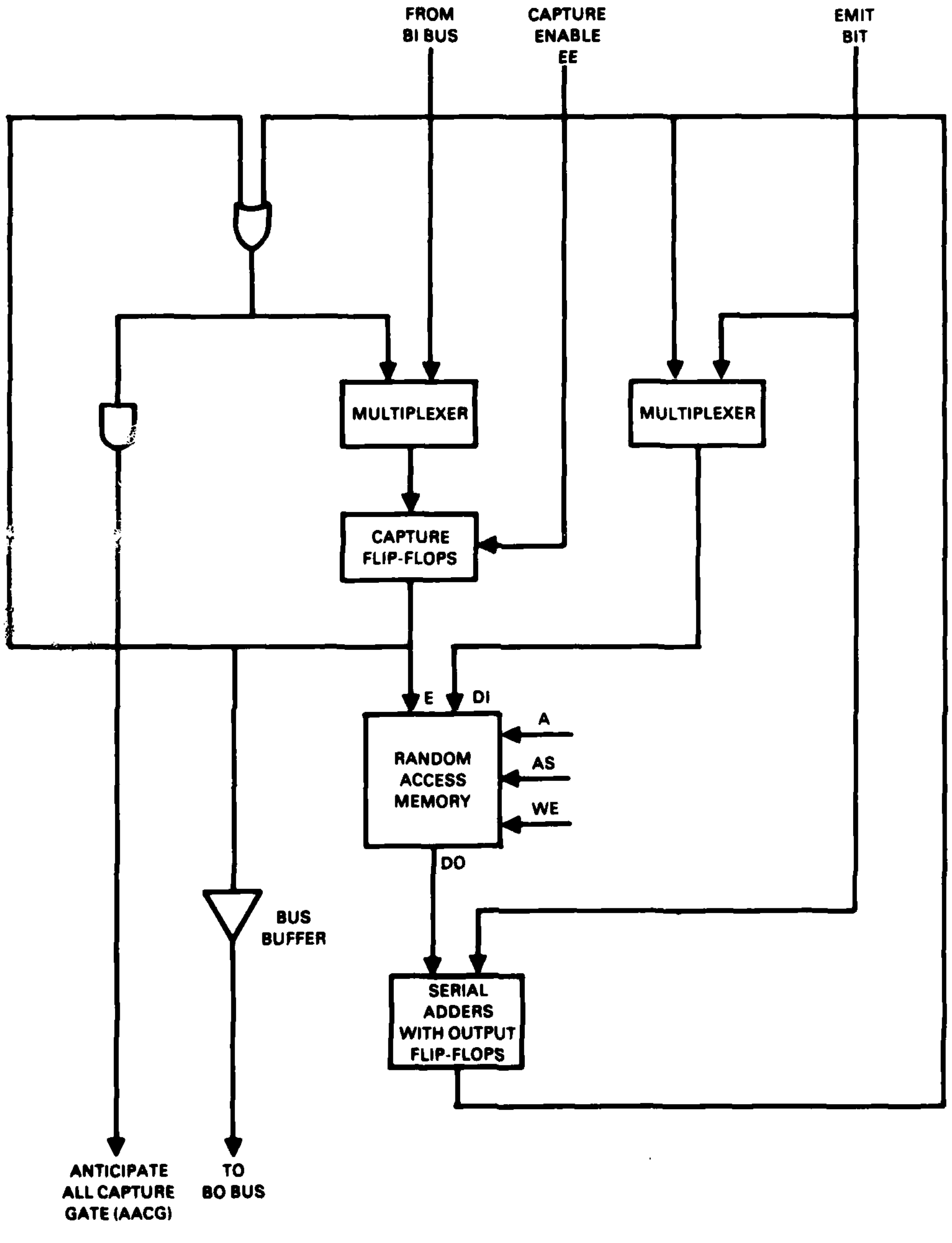


Figure 5. Associated memory logic bank.

using a separate memory component for each bit position. Note that each bit position has an independent data input (DI), data output (DO), write enable (WE), and general enable (E). All the bits of the word share the same address input (A) and a common general enable (AS). The write enable (WE) and both general enables (E and AS) must all be true for data to be written. The write enable must be false and both general enables must be true for data to be read. When these conditions are not met, the data output (DO) is a one.

The RAM data output (DO) associated with each bit position is connected to one input of a serial adder/subtractor associated with that bit position. The other input of all these serial adder/subtractors is connected to a common signal called EMIT. Each serial adder contains an output register for synchronization of its output with the system clock.

The RAM data inputs are connected to the output of a multiplexer. The multiplexer selects, for all memory word bit positions simultaneously, either the output of the respective serial adder/subtractor associated with each bit position or a common input of EMIT.

The general enable from each RAM bit position (E) is connected to the output of a flip-flop called a Capture Flip-Flop. When the common general enable (AS) is true, the presence of zero or one in a Capture Flip-Flop enables or disables the respective associated memory bit position. When a memory bit position is disabled, attempts to write into the memory bit position are ignored and attempts to read result in a memory bit position data output (DO) of one regardless of the memory contents. The Capture Flip-Flops are clocked whenever the common Capture Flip-Flop enable input (EE) is true.

The data inputs of the Capture Flip-Flops are connected to the output of another multiplexer. This multiplexer selects, for all Capture Flip-Flops simultaneously, either the output of the serial adder/subtractor ORed with the previous output of the Capture Flip-Flop for the respective bit position, an independent input (BI) for the respective bit position, or zero. The first of these selections permits the occurrence of a one generated by the respective serial adder/subtractor to be "captured," that is, to remain in the Capture Flip-Flop for succeeding clock cycles. The other multiplexer selections are used to initialize the Capture Flip-Flops.

The outputs of the Capture Flip-Flops are connected to two additional places. They connect to a gate, called the Anticipate All Capture Gate, whose output (AACG) is true if and only if all the Capture Flip-Flops will be in the one state if the Capture Flip-Flops are clocked. They also connect to bus buffers permitting their output to be logically connected or disconnected from a bus (BO). The bus consists of independent signals for each respective memory bit position.

As shown in figure 6, the Associative Memory Logic is partitioned into identical banks. Each bank contains a segment of the total memory word addressed by the common address lines (A). Two busses, one for input (BI) and the other for output (BO), interconnect the banks. The general enable input (AS) and the Anticipate All Capture Gate output (AACG) for each bank remain independent, thus permitting the banks to be selected and examined individually.

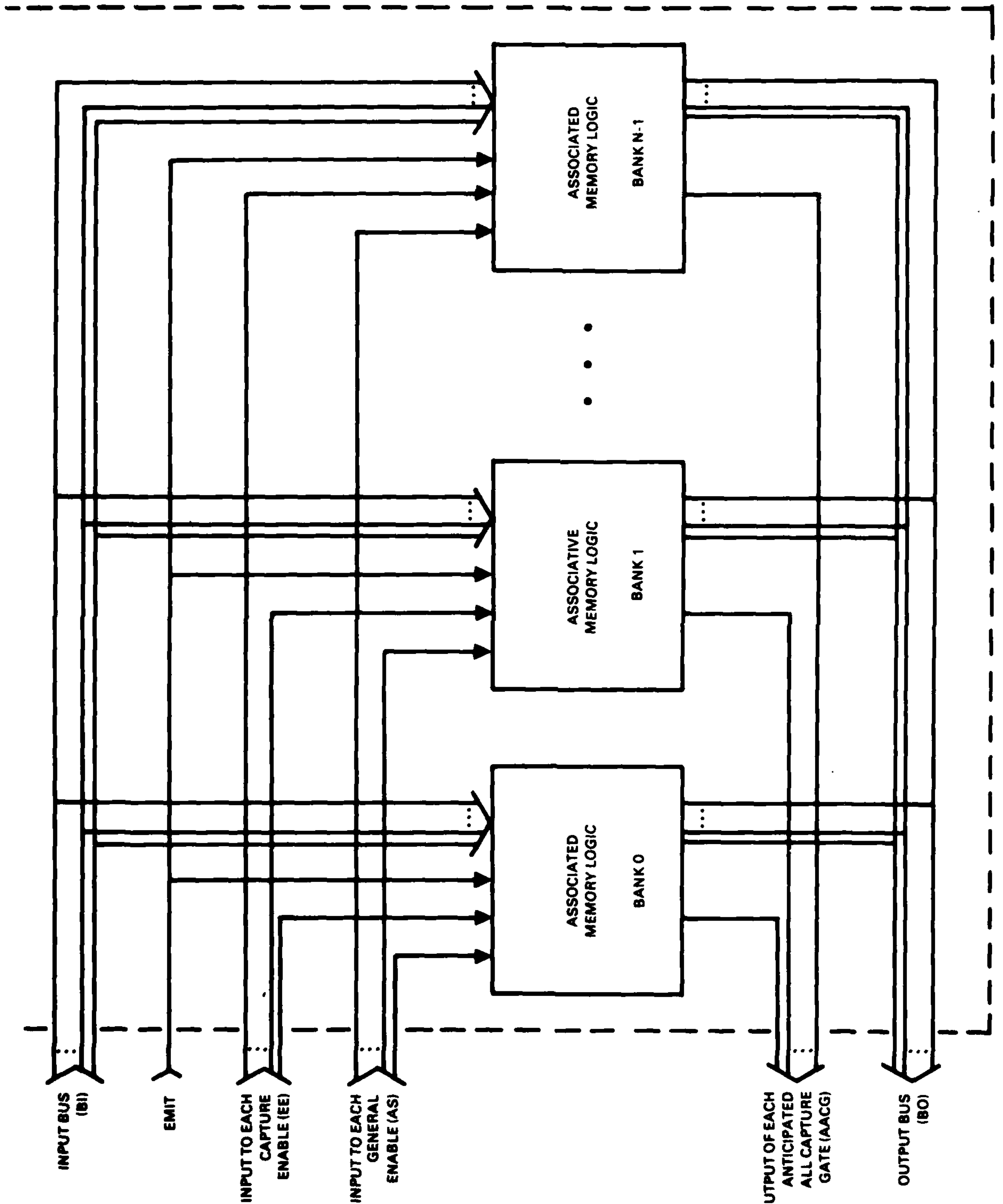


Figure 6. Associated memory logic.

5.2.2 Memory Selection Logic

The Memory Selection Logic interconnects the banks of Associated Memory via the Input and Output Busses (BI and BO) and the independent signals (AS, BE, EE, and AACG) of each Associative Memory Logic bank. A block diagram of the Memory Selection Logic is shown in figure 7. The Memory Selection Logic consists of the Bank Selection Logic and the Bit Selection Logic. The Bank Selection Logic is used to control the activation of the Associative Memory Logic banks by generating an independent general enable (AS) for each bank. It also activates one of the independent Output Bus buffer enables (BE) to logically connect the output of a single bank which it selects to the Bit Selection Logic. Similarly, the Bit Selection Logic controls the activity of the bit positions within each bank.

Internally, the Bank Selection Logic contains a Bank Capture Flip-Flop for each bank. The output of each Bank Capture Flip-Flop is connected to a respective bank general enable (AS). As in the case of the Associative Memory Logic Capture Flip-Flops, contents of zero and one are defined as enable and disable, respectively. A particular bit position within the Associative Memory Logic word is enabled only if both its Associative Memory Logic Capture Flip-Flop and its Bank Capture Flip-Flop contain zero.

A multiplexer supplies the input to the Bank Capture Flip-Flops. It selects either the Bank Priority Logic output, the Bank Decoder output, or zero.

The Bank Priority Logic is used to determine the highest-priority bank containing at least one Associative Memory Logic Capture Flip-Flop in the zero state. The inputs to the Bank Priority Logic are the independent outputs of the Anticipate All Capture Gates (AACGs) from each Associative Memory Logic bank. Both encoded and decoded priority output are provided by the Bank Priority Logic. An eight-bit gate-level implementation is shown in figure 8. The encoded output provides the most-significant input to the Priority Register, which is located within the Register Logic and is the subject of the next section. The Bank Capture Flip-Flop input multiplexer uses the decoded output. The decoded output also provides independent Output Bus buffer enables (BEs) for each Associative Memory Logic bank.

The Bank Decoder and zero multiplexer selections are used to initialize the Bank Capture Flip-Flops. All Bank Decoder outputs are ones except the output addressed by a field of the Exchange Register. The Exchange Register is part of the Register Logic which is discussed in the next section. This path permits the contents of the Exchange Register to address a single bank directly by loading all the Bank Capture Flip-Flops with ones except the bank being addressed.

In addition to the Bank Selection Logic, the Memory Selection Logic also contains the Bit Selection Logic, which is used to control the activity of the bit positions within the Associated Memory Logic banks. This is accomplished through the input (BI) and output (BO) busses which interconnect all the banks in parallel.

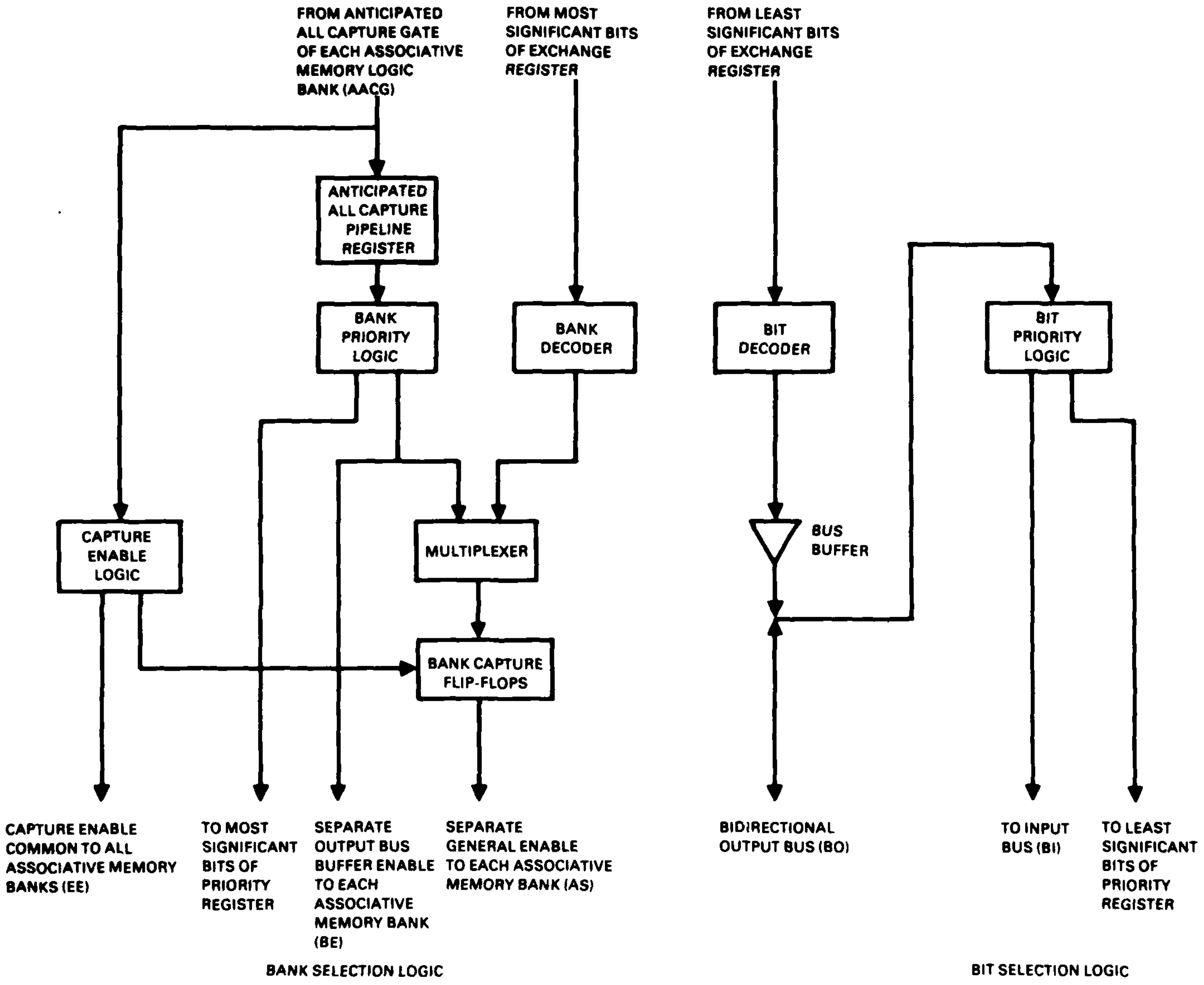

FROM ANTICIPATED ALL CAPTURE GATE OF EACH ASSOCIATIVE MEMORY LOGIC BANK (AACG)
FROM MOST SIGNIFICANT BITS OF EXCHANGE REGISTER
FROM LEAST SIGNIFICANT BITS OF EXCHANGE REGISTER
ANTICIPATED ALL CAPTURE PIPELINE REGISTER
BANK PRIORITY LOGIC
BANK DECODER
BIT DECODER
BIT PRIORITY LOGIC
CAPTURE ENABLE LOGIC
MULTIPLEXER
BUS BUFFER
BANK CAPTURE FLIP-FLOPS
CAPTURE ENABLE COMMON TO ALL ASSOCIATIVE MEMORY BANKS (EE)
TO MOST SIGNIFICANT BITS OF PRIORITY REGISTER
SEPARATE OUTPUT BUS BUFFER ENABLE TO EACH ASSOCIATIVE MEMORY BANK (BE)
SEPARATE GENERAL ENABLE TO EACH ASSOCIATIVE MEMORY BANK (AS)
BIDIRECTIONAL OUTPUT BUS (BO)
TO INPUT BUS (BI)
TO LEAST SIGNIFICANT BITS OF PRIORITY REGISTER
BANK SELECTION LOGIC
BIT SELECTION LOGIC

Figure 7. Memory selection logic.

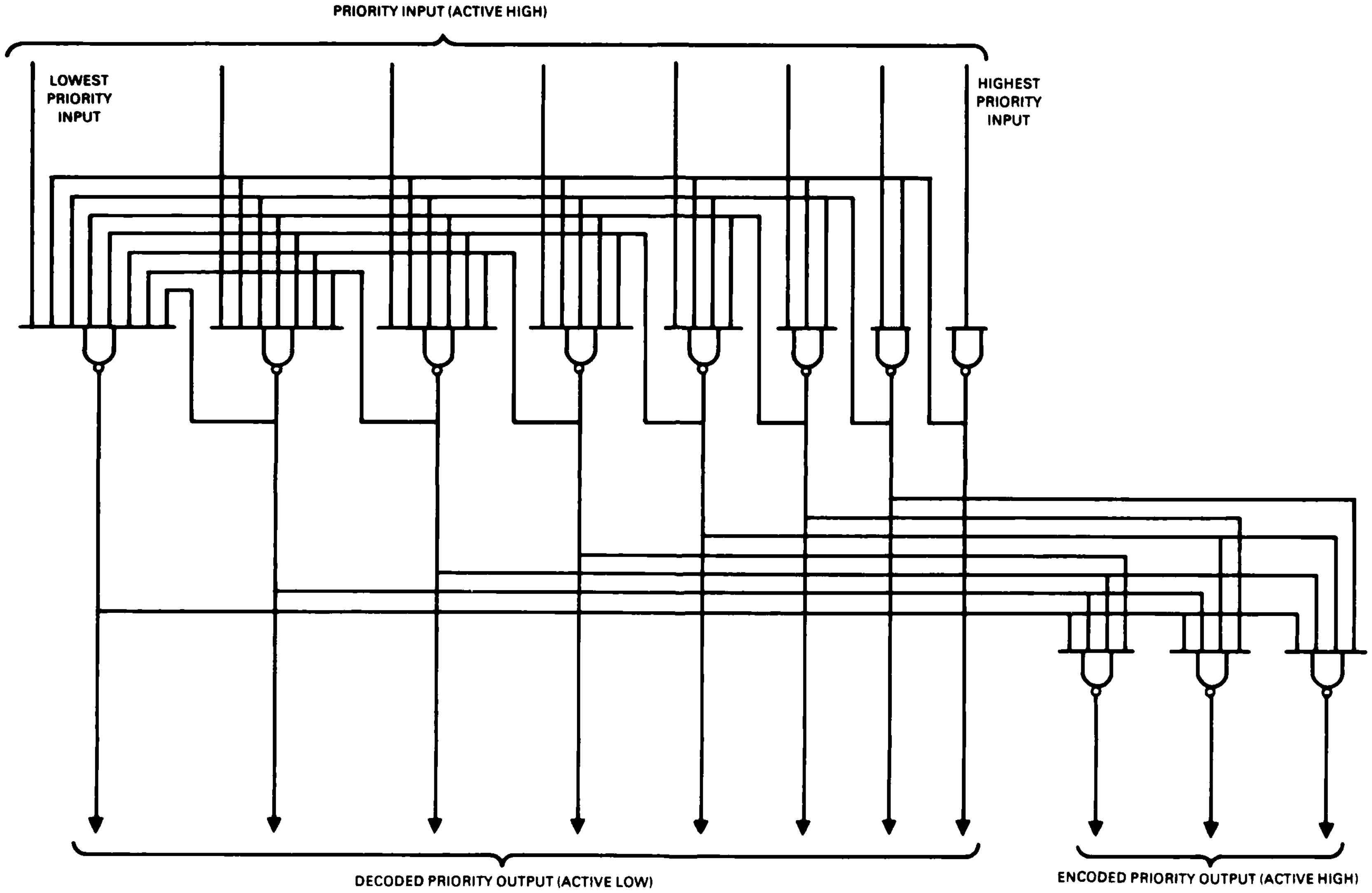

PRIORITY INPUT (ACTIVE HIGH)
LOWEST PRIORITY INPUT
HIGHEST PRIORITY INPUT
DECODED PRIORITY OUTPUT (ACTIVE LOW)
ENCODED PRIORITY OUTPUT (ACTIVE HIGH)

Figure 8. 8-bit priority circuit.

The Bit Selection Logic contains a Bit Decoder driven by the Exchange Register of the Register Logic. The output of the Bit Decoder is logically connected to the Output Bus (BO) through its own bus buffer. The buffer selected by the Bank Selection Logic is forced to disconnect when the Bit Decoder buffer is enabled. The Bit Decoder works in conjunction with the Bank Decoder so that a single bit position within a single bank can be addressed by the Exchange Register.

The Bit Selection Logic also contains Bit Priority Logic. This is designed in the same manner as the Bank Priority Logic. The Bit Priority Logic prioritizes the bits found on the Output Bus (BO). Its decoded output directly drives the Input Bus (BI). Its encoded output generates the least-significant input to the Priority Register within the Register Logic.

Functionally, it seems more appropriate to logically connect the Bit Decoder to the Input Bus (BI) rather than to the Output Bus (BO), since its final destination is the Input Bus inputs of the Associative Memory Logic. By logically connecting the Bit Decoder to the Output Bus (BO), the same result is produced, since the decoded output of the Bit Priority Logic will echo its input when only one bit position is activated. The Output Bus was chosen because the Bit Priority Logic is in a critical timing path.

5.2.3 Register Logic

The Register Logic encompasses the interface to the host processor, the Exchange Register, the Priority Register, the Sub-Priority Counter, the Real-Time Counter, and the EMIT Generation Logic. A block diagram is shown in figure 9.

The Exchange Register holds the data word received from the host processor during its write operation. Depending on the function invoked, the contents of the Exchange Register may then be transferred to the Clock Counter or be used in serial form by the Associative Memory Logic through the EMIT Generation Logic. The Exchange Register holds the data output to be sent to the host processor during its read operation. Depending on the function invoked, the contents of the Exchange Register may originate from the Clock Counter, the Priority Register, or the Associative Memory Logic. In the latter case, the Exchange Register operates as a shift register to collect the bits generated by the Associative Memory Logic.

The Priority Register is a register which maintains the priority-encoded result of the last prioritization of the Memory Capture Flip-Flops. Input to the most- and least-significant halves of the Priority Register is generated by the Bank Priority Logic and Bit Priority Logic portions of the Memory Select Logic, respectively.

The Sub-Priority Counter is an up-down counter used with the priority queue algorithm. Its contents can be incremented, decremented, or initialized with the contents of the Exchange Register.

The Real-Time Counter is used to maintain the current time. Its contents can be incremented or initialized with the contents of the Exchange Register. The Real-Time Counter is partitioned into most- and least-significant portions for greater time interval range. Loading and incrementing of the Real-Time

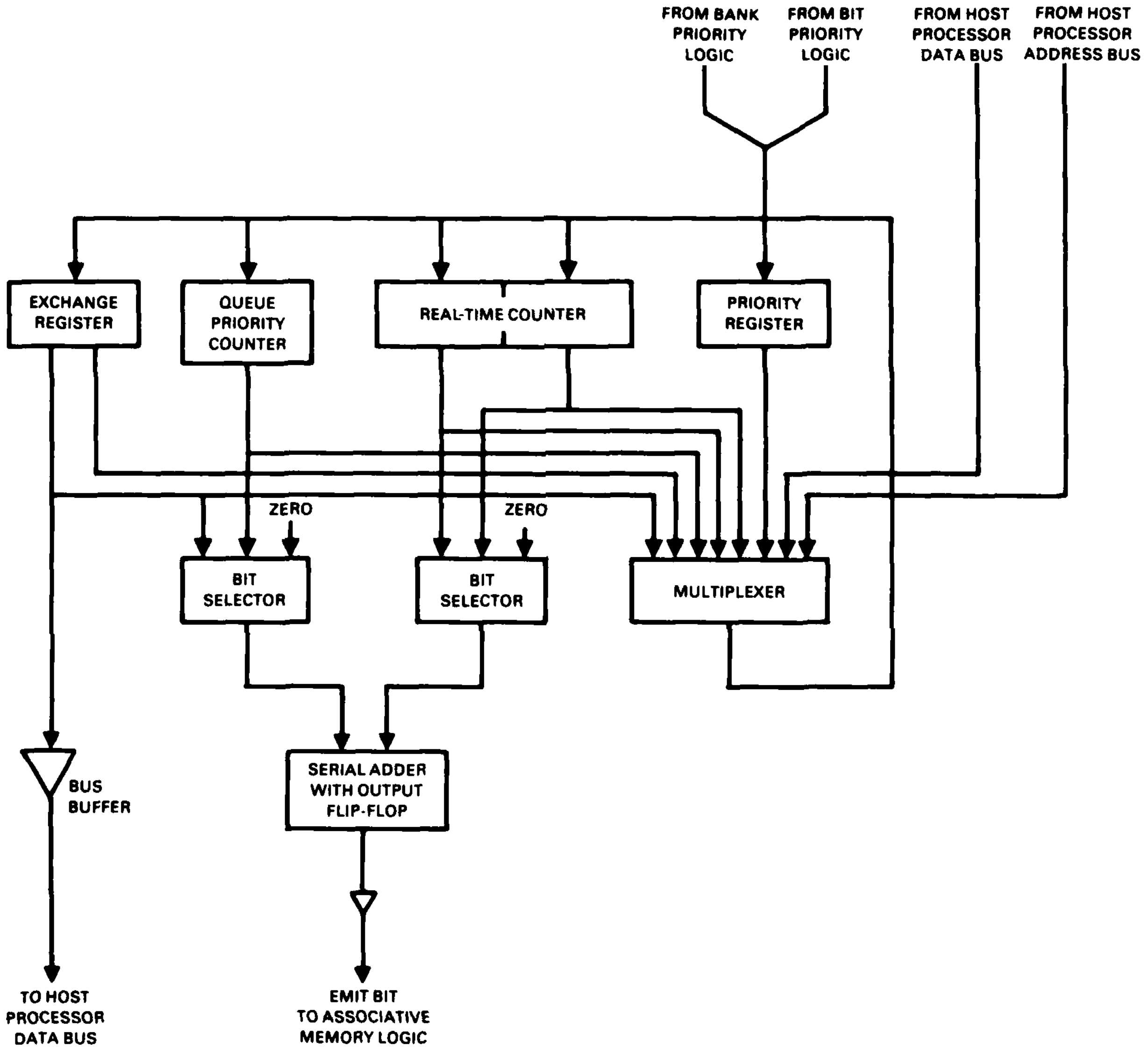


Figure 9. Register logic.

Counter is under the control of the Microsequence Control Logic to ensure proper synchronization with its algorithms.

The EMIT bit, the output of the EMIT Generation Logic, is used by the Associative Memory Logic as a common data input to all its serial adder/subtractors and a common data input to all its memory data input multiplexers. The EMIT Generation Logic consists of a serial adder and two multiplexers, called bit selectors, which feed its inputs. One bit selector is wired to select any of the Exchange Register bits, any of the Sub-Priority Counter bits, or zero. The other bit selector is wired to select any of the Real-Time Counter bits or zero. The multiplexers are wired such that, if both multiplexers select from a register or counter rather than zero, the same bit position is accessed. The output of the serial adder is the EMIT bit.

5.2.4 Microsequence Control Logic

The Microsequence Control Logic generates the control signals for the rest of the logic of the Hardware Executive processor. It contains the clock oscillator, the host processor interface control logic, microprogram memory, and logic to generate the next microinstruction address. A block diagram of the Microsequence Control Logic is shown in figure 10.

The circuitry that generates the microinstruction addresses for all microinstruction cycles is called the Transform Logic. Usually, the Transform Logic obtains the bits constituting the next microinstruction address from a field in the microinstruction. There is no counter circuit acting as a program counter. The Transform Logic is used to convert the address presented by the host processor into a microprogram entry address. The signals indicating whether the host access is a read or a write are used by the Transform Logic in conjunction with the host-provided address to generate unique microprogram entry addresses for each combination within the Hardware Executive address space. The Transform Logic also performs conditional jumps on the state of the Anticipated All Captured signal, the logical OR of the individual Anticipated All Capture Gate outputs (AACGs) from the Associative Memory Logic banks.

5.3 EXECUTIVE FUNCTION INTERNAL OPERATION

This section presents the algorithms used to implement various executive functions on the hardware described above. A complete summary of executive functions is given in appendix A.

5.3.1 Table Organization

The Task and Event Tables are stored in an associative memory. The associative memory is implemented from conventional random access memory (RAM). The word at a particular physical RAM address contains a bit for the same bit position of every entry provided by the host processor. One way to visualize this organization is to view the associative memory as a two-dimensional matrix of bits where segments of rows correspond to words mapped into the host processor address space and columns correspond to words of physical RAM. When the host writes directly into the tables, the microprogram of the Hardware Executive must serially load the word from the host a bit at a time into successive physical RAM addresses. Similarly, when the host reads directly from

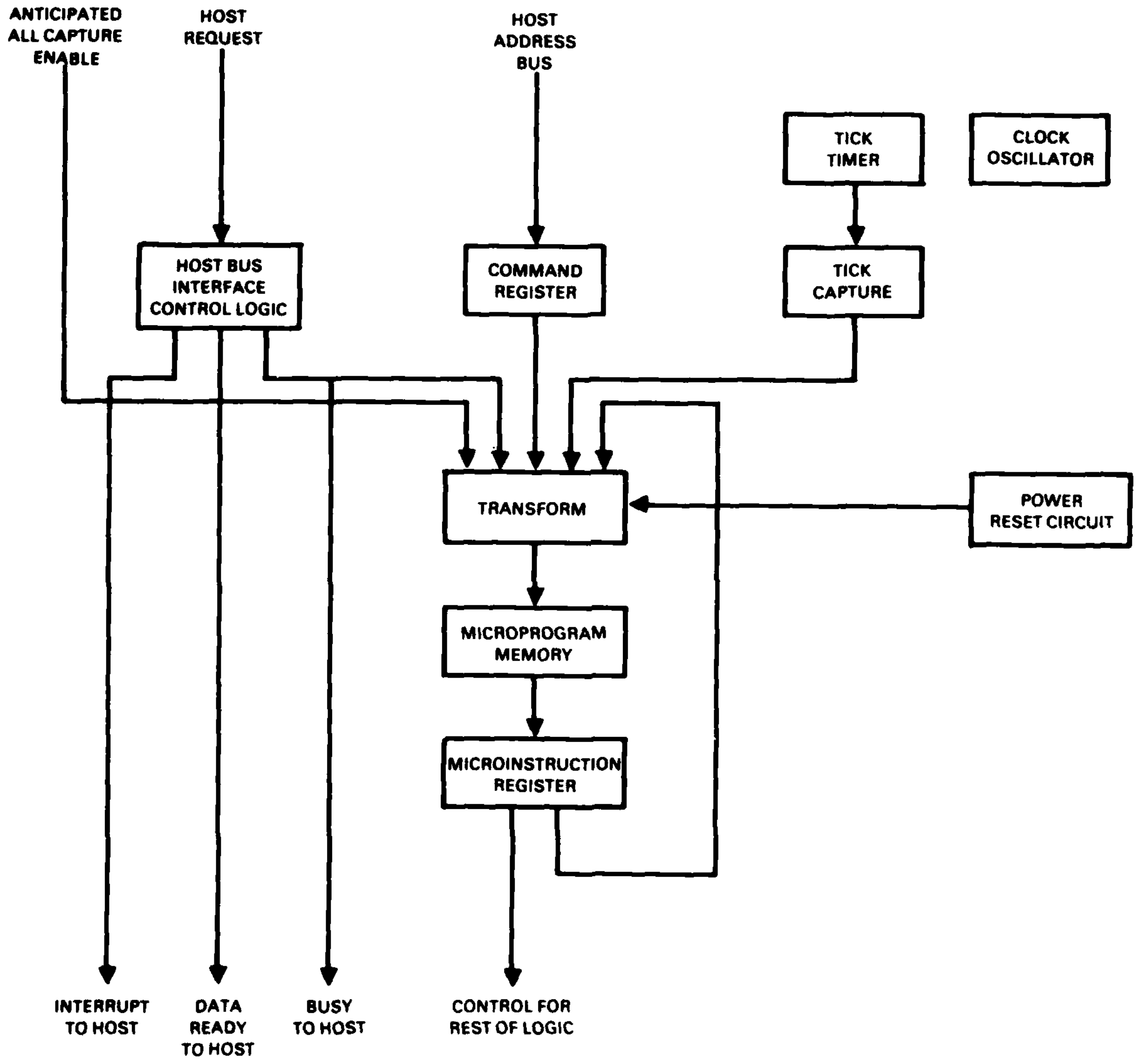


Figure 10. Microsequence control logic.

the tables, the operand read must be collected serially from successive physical RAM addresses. The format of the task and event tables is presented in appendix B.

Task Table entries are logically divided into six fields. The first, consisting of a single bit called the Empty Bit, indicates whether the table location is empty or contains valid table data. There are two priority fields. The User Priority field is loaded by the host processor. The Queue Priority field is maintained by the Hardware Executive microprogram. It is used to order tasks on a first-come-first-served basis within a given user priority. The Task State field contains three bits to indicate one of five possible states: Suspended-Waiting, Suspended-Ready, Waiting, Ready, and Running. When the task is waiting on a semaphore, the Wait Semaphore field contains the associated Semaphore Identifier. The last field is the Processor field. It is only required for multiprocessor applications when a common Hardware Executive is used. It indicates which processor is taking custody of a Task.

Normal Event Table entries have five fields. The Empty Bit and Processor fields serve the same functions as above. The Normal Event Registration field contains the event registration hierarchy information. The Event State field contains one bit to indicate whether the event is Armed or Triggered. Finally, the Signal Semaphore field contains the Semaphore Identifier of the Semaphore to be signalled when the event is triggered. Entries of the Time Event Table have the same format as those of the Normal Event Table except that the Event field is replaced with the absolute time the event is to be triggered.

5.3.2 INITIALIZE Functions

The Empty Bit of the task and event table locations indicates whether the location contains relevant data or is empty. Values of zero and one represent allocated and empty, respectively. Execution of the INITIALIZE function establishes all table locations as empty by setting their respective Empty Bits. Since the random access memory (RAM) components implementing the associative memory access the same bit position of all table locations simultaneously, the setting of all Empty Bits is accomplished with a single access cycle for each table. The INITIALIZE function also clears the Task Queue Priority Counter of the Register Logic.

5.3.3 Reserve Task and Event Functions

The RESERVE TASK function searches for an empty task table location and returns its index to the host processor if found. It also clears the Empty Bit and sets the Task State field to Suspended-Ready within the entry location selected. A negative index returned to the host processor indicates that no task table locations are available.

The microprogram performs the search by first addressing the physical RAM word containing all the Empty Bits. If the bit is a one, then the associated table entry is defined as empty. Each bit is complemented by its adder/subtractor and then loaded into its respective Associative Memory Capture Flip-Flop. On the next microinstruction cycle, the Anticipated All Captured Gate (AACG) outputs from the Associative Memory Logic banks are prioritized by the Bank Priority Logic. At this point, the Bank Priority Logic enables the

Associative Memory Logic Output Bus Buffer of the highest-priority bank containing at least one empty location. The contents of the Output Bus are then prioritized by the Bit Priority Logic to choose the highest priority empty location. The encoded output of the Bank and Bit Priority Logic is loaded into the Priority Register at the end of the cycle. The next cycle moves the Priority Register contents to the Exchange Register. At this point, the Exchange Register contains the index of the location to be returned to the host processor. It is also routed through the Bank and Bit Decoders to set the capture flip-flops so that only the entry to be allocated is enabled. Finally, the Empty Bit of the enabled entry is cleared, and the Task State field is initialized to Suspended-Ready. The algorithm is the same for the Event Table mutatis mutandis.

5.3.4 SUSPEND and RESUME Task Functions

The SUSPEND function is designed to exclude a task from further dispatch. If the task is in the Ready or the Running task states, the SUSPEND function moves it to the Suspended-Ready task state. If the task is in the Waiting task state, the SUSPEND function moves it to Suspended-Waiting task state. The RESUME function reverses the process. The Suspend Bit, one of the bits of the Task State field, is dedicated to indicating whether the task is suspended or not. When the Suspend Bit contains a one, the task is suspended and hence is not included in dispatch checks.

First, the SUSPEND and RESUME functions must, respectively, set and clear the Suspend Bit. The functions are implemented by loading the data supplied by the host processor into the Exchange Register, then loading the output of the Bank and Bit Decoders into the Associative Memory Capture Flip-Flops to enable a single task table entry, and finally writing the appropriate value into the Suspend Bit of the selected entry.

When the RESUME function is applied to a task in the Suspended-Ready task state, the task enters the Ready task state. Any time a task enters the Ready task state from one of the other task states, the contents of the Task Queue Priority Counter must be copied into the Task Queue Priority field of the associated task table entry. The contents of the Task Queue Priority Counter are then incremented. This enables succeeding tasks entering the Ready task state to have progressively higher numeric subpriorities.

When the SUSPEND function is applied to a task in the Ready or Running task states, the task enters the Suspended-Ready state. Any time a task moves from the Ready or Running states to some other state, the Task Queue Priority fields of all tasks with numerically higher subpriorities and the Task Queue Priority Counter must be decremented. This prevents subpriority overflow.

5.3.5 DISPATCH CHECK Function

The DISPATCH CHECK function returns to the host processor the task table index of the task with the highest Task Priority whose Task State is either Ready or Running. A negative result is returned if no task qualifies.

Task Priority should not be confused with the Bank and Bit Priority Logic of the hardware implementation. The Task Priority consists of the contents of the Task User Priority field and the Task Queue Priority field. The content of the Task User Priority field is supplied solely by the host processor which writes directly into the Associative Memory address space during task creation. The content of the Task Queue Priority field is supplied by the Queue Priority Counter of the Register Logic and is under the control of the microprogram. The Bank and Bit Priority Logic serves an entirely different function. *It selects a single table entry from all the table entries whose* Associative Memory Capture Flip-Flops contain zero. It thus provides a way to select a unique table entry when several satisfy the same given criteria.

The DISPATCH CHECK function first initializes all the Capture Flip-Flops to zero. One of the microprogrammable functions of the Associative Memory Capture Flip-Flops causes them to be set to one by an output of one from their respective Associative Memory entries. Once set, however, they remain set throughout the continuous use of the same function during the succeeding microinstruction cycles. This capture microinstruction function is first applied to the task table Empty Bit. Only those task table locations whose Empty Bits are zero, indicating that the table locations contain a table entry, will have zero in their respective Associative Memory Capture Flip-Flops at the end of the cycle. This capture microinstruction is next applied to the Suspend Bit of the Task State field. At the end of this cycle, only those task table locations for whom both the Empty Bit and the Suspend Bit are zero have zero in their respective Associative Memory Capture Flip-Flops. The same technique is next applied to the Wait Bit of the Task State field. The Wait Bit is one if the task is waiting on a semaphore and zero otherwise. The strategy of each cycle is to further reduce the list of possible task table locations qualifying for dispatch by setting the Capture Flip-Flops of those which fail each test.

The highest-priority task is defined as the task having the lowest numeric value in its User and Queue Priority fields. The bits of the User and Queue Priority fields are examined sequentially, starting with the most-significant bit and proceeding to successively less-significant bits of first the Task User Priority field and then the Task Queue Priority field. The Associative Memory Capture Flip-Flops are set by ones emanating from the User and Queue Priority field bit positions of each respective entry only if so doing does not cause all the Associative Memory Capture Flip-Flops to contain ones. Setting of the Capture Flip-Flops is inhibited by logically gating the Anticipated All Capture Gate (AACG) outputs of the Associative Memory banks together and logically connecting the result to the Associative Memory Capture Flip-Flop enable (EE). The output of the Bank and Bit Priority Logic is stored in the Priority Register on each cycle. After all these bit positions have been examined, the Priority Register contains the entry location index corresponding to the task to be dispatched. Finally, the contents of the Priority are moved to the Exchange Register for return to the host processor.

5.3.6 SIGNAL and WAIT Functions

The SIGNAL and WAIT functions permit separate tasks to coordinate their utilization of common resources. A task requests access to a resource with the WAIT function. If the resource is available, the task is given access. Otherwise, the task is moved to the Waiting task state. A task releases claim to a resource by using the SIGNAL function.

Many executives have a flag associated with each resource, called a binary semaphore, to indicate whether the resource is available. There is no physical bit internal to the Hardware Executive representing the semaphore. Availability of a resource is implicitly known from the contents of the task table Wait Semaphore fields. If no task in the Waiting task state has a particular Semaphore Identifier in its Wait Semaphore field, then the semaphore is in the available state.

The SIGNAL function searches for the highest-priority nonsuspended task whose Wait Semaphore field matches the Semaphore Identifier supplied by the host processor. Once found, the task is placed in the Ready task state, its Task Queue Priority field is loaded with the contents of the Task Queue Priority Counter, and the Task Queue Priority Counter is incremented. Finally, the Wait Semaphore field is cleared. The SIGNAL function should be followed by a DISPATCH CHECK function since the task placed in the Ready task state may have a higher priority than the task currently in the Running task state.

The WAIT function first examines the Wait Semaphore fields of all nonsuspended tasks for a match with the Semaphore Identifier supplied by the host processor. If none is found, the currently running task can continue to run, since the resource is available. Otherwise, the task is moved to the Waiting task state and the queue priorities of all tasks with a higher numeric Task Queue Priority field are decremented. The Task Queue Priority Counter is also decremented. In either case, the Semaphore Identifier is loaded into the Wait Semaphore field. The WAIT function is followed by a DISPATCH CHECK function since the host processor does not know if the currently running task can continue running.

The identification of all tasks whose Wait Semaphore fields match the Semaphore Identifier supplied by the host processor is accomplished as follows. The Semaphore Identifier is located in the Exchange Register. Bit Selection Logic permits the individual bits of the Exchange Register to be fed to the Associative Memory Logic through the common EMIT signal. One microinstruction cycle is executed for each bit position of the Wait Semaphore field. The EMIT signal, containing a bit of the Semaphore Identifier, is subtracted in parallel from each Associative Memory RAM output, containing a bit of the Wait Semaphore field for the respective table entry, by the Associative Memory Logic adder/subtractors. If the Semaphore Identifier is numerically equal to the Wait Semaphore field, the subtracted result is a sequence of zeros, one for each bit position of the Wait Semaphore field. If they are not numerically equal, at least one of the bits in the sequence is a one. The capture function of the Associative Memory Capture Flip-Flops can be used to capture the occurrence of any ones. Those Capture Flip-Flops still containing zeros after all the bits of the sequence have been applied correspond to table entries where a match exists.

5.3.7 Events

An event is defined as a "hardware- or software-initiated perturbation" [8] that causes the executive to transfer control to special user-defined processing. The event concept may be viewed as a generalization of the hardware interrupt concept which includes pseudointerrupts initiated entirely by software. An event is triggered from software by invoking a cause event function of the executive, with the event to be triggered passed as a parameter.

Events caused by hardware interrupts are implemented by coding the cause event function within the routine where hardware transfers control.

The Hardware Executive internally distinguishes between two types of events, normal events and time events. Normal events are triggered explicitly by the user through the CAUSE NORMAL EVENT function. They are identified by an Event Identifier, the parameter passed to the CAUSE NORMAL EVENT function. Time events are triggered by the expiration of a time interval. They can be visualized as user-settable alarm clocks. Time events are identified by their trigger time, an absolute time value on the Real-Time Clock internal to the Hardware Executive.

Events are registered within the Hardware Executive by first obtaining an event table location with the RESERVE NORMAL EVENT or RESERVE TIME EVENT function. Then the host processor directly loads the event table entry with the event definition. The processing associated with an event is defined as a task waiting on a semaphore. The host processor loads the Signal Semaphore field with the Semaphore Identifier of the semaphore to be signalled when the event is triggered. Several events may map into the same Semaphore Identifier if desired. The Event State field is initialized as Armed.

5.3.8 CAUSE NORMAL EVENT Function

Normal events are organized into an event hierarchy. The Event Identifier is divided into subfields, each subfield corresponding to one level of the hierarchy. The most-significant subfield represents the top of the hierarchy, the general class of event. The next most-significant subfield represents the subclass within the class represented by the most-significant subfield, and so on. The hierarchy permits the registration of default event at each level in the hierarchy. A default event is triggered when the event supplied by the user is not registered but the events class, subclass, subsubclass, etc, are defined to the level of the default definition.

When the CAUSE NORMAL EVENT function is invoked, a search is first made to determine whether an exact match exists between the given Event Identifier and the nondefault event registrations in the event table. If none is found, the search is repeated for the default event of the same level in the hierarchy as the given event. If that fails, the search is repeated for the default event on the next highest level of the hierarchy. This process continues until the top of the hierarchy is reached. If there is no matching default definition at the top, no event is triggered.

The CAUSE NORMAL EVENT function should be followed by a NORMAL EVENT SIGNAL CHECK function to signal the semaphore associated with the newly triggered event.

5.3.9 Time Event Triggering

The Real-Time Clock is not driven directly by a time-base oscillator. The time-base oscillator generates a pulse which sets a flip-flop called the Tick State Flip-Flop. The output of the Tick State Flip-Flop is wired into the Transform of the Microsequence Control Logic. When the microprogram branches to the address generated by the Transform, it will automatically branch to a unique routine to process a Real-Time Clock tick when the Tick

Flip-Flop is set. This routine increments the Real-Time Clock within the Register Logic and clears the Tick Flip-Flop. It then compares the value of the Real-Time Clock with the trigger time values of all the Time Events in parallel. The event states of all time events whose trigger time value equals the Real-Time Clock value are changed to the Triggered state.

5.3.10 Event Signal Check Functions

The processing associated with an event is defined as a task waiting on a semaphore. The NORMAL EVENT SIGNAL CHECK and TIME EVENT SIGNAL CHECK functions are used to signal the semaphore corresponding to the highest priority triggered event. These functions move the event from the Armed state to the Triggered state when they signal their associated semaphore. They return a negative value to the host processor if no event is found in the Triggered state. The NORMAL EVENT SIGNAL CHECK and TIME EVENT SIGNAL CHECK functions should be followed by a DISPATCH CHECK function to identify the task that was waiting on the semaphore signalled.

5.4 Circuit Design Considerations

This section presents various issues relating to the hardware design of the Hardware Executive.

5.4.1 Associative Logic Design Alternatives

The design of the associative memory logic must permit the microprogram instructions to address the same bit position of all table entries simultaneously. The design also must permit the modification of a single bit within a single entry without affecting the rest of the table entries. Both requirements are satisfied by utilizing random access memory (RAM) integrated circuits with only one bit of data input and output. The component address inputs are connected to the table entry bit position field of the microinstruction. Since there is one component for each table location, a separate write enable per location permits modification of a unique bit.

The physical RAM implementing the associative memory would not need individual integrated circuits for each bit position of the parallel accessed word if a memory with independent write enables for each bit position were available. Unfortunately, no off-the-shelf integrated circuit components with this feature have been found. Another approach is to use a component with only a single write enable, to read the output word of the component into a latch, to modify only the desired bits in the latch, and then to write the contents of the latch back into the memory. This approach saves power since the address decoding logic inside each memory component is not replicated for each output bit and because one can take advantage of a wider selection of available low-power components. On the other hand, the approach requires more complex logic and a longer microinstruction cycle time.

5.4.2 Size of Executive Tables

The maximum number of task or event registrations processed simultaneously within a single Hardware Executive module is equal to the number of associative memory locations, called elements, which are processed in parallel. The SDEX/20 Version Log was examined to aid in the determination of an appropriate maximum per module. SDEX/20 [14] is the predecessor standard executive of SDEX/M. The log contains the values assigned to the compile time parameters for each released version. As of 21 October 1979, only 42 out of 60 compiled versions are recorded in the log as delivered to users and are considered in this study. Statistics were generated from the sizes of the version task tables. SDEX/20 versions have from one to four task types, each type having its own task table. The percentage of versions covered by associative memories of various sizes are tabulated below.

Maximum Table Size	16	32	64	128
All Tasks	5 %	29 %	57 %	95 %
Successor Tasks	29 %	57 %	88 %	100 %
Time-Dependent Tasks	79 %	98 %	100 %	100 %
Time-Critical Tasks	83 %	98 %	100 %	100 %
Background Tasks	93 %	98 %	100 %	100 %

Considering all task types, 64 elements adequately cover the majority of SDEX/20 versions, and 128 cover nearly all versions.

The quantity of hardware required to implement associative memories of various sizes must also be considered. The table below tabulates the number of integrated circuits and the power consumption for a typical TTL implementation of the associative logic.

Maximum Table Size	16	32	64	128
Integrated Circuits	46	92	184	368
Typical Power (watts)	5.96	11.92	23.85	47.69
Maximum Power (watts)	8.55	17.09	34.18	68.36

where each 8-bit slice (bank) is implemented from:

2 25LS15 quad serial adder/subtractors
4 25LS158 2-to-1 inverting multiplexers
2 25LS258 2-to-1 inverting three-state multiplexers
8 27LS01 256-word-by-1-bit random access memories
1 74LS00 quad 2-input NAND gate (address buffers)
1 74LS04 hex inverters (address buffers)
1 74S30 8-input NAND gate (all captured detect)
2 74LS32 2-input OR gates (capture loop)
2 74LS398 4-bit 2-to-1 multiplexer register.

The size selected will depend on the mechanical and electrical constraints of the desired form factor. The demonstration breadboard contains 64 elements.

[14] NAVELEX User's Handbook: AN/UYK-20(V) Computer Support Software, NAVELEX 0967-LP-598-2020, v 2, change 6, Oct 1978.

Examining the components individually shows that 35 percent of the integrated circuits and 74 percent of the typical power consumption are associated with the 256-word-by-1-bit random access memories (Am27LS01s) implementing the associative memory. A large number of memory component data sheets were examined in a variety of technologies before selection. Thus selection of the memory component is a critical design consideration.

5.4.3 Executive Table Expansion

The maximum number of executive table entries could be expanded within the same Hardware Executive module by constructing two or more tables in the associative memory and manipulating them sequentially [15]. In such a scheme, some of the memory component address lines would select the table while others would select the bit position of an entry within the selected table. Locating unused space for the additional tables in the memory components implementing the associative memory is not a problem, since their capacity is far greater than required. The design presented in this paper used fewer than half of the 256 bits available in the 27LS01. The NMOS 2125 or the CMOS 6508 are 1024-bit integrated circuits which could be used if more space is required.

Another approach to executive table expansion is to provide additional Hardware Executive modules for parallel operation. The priority logic of the modules must be interconnected for the ensemble to generate a unique result.

The first method has the advantage of reducing the quantity of parallel associative logic. The second method has the advantage of simpler microprogramming and faster execution. The second method is recommended because the Hardware Executive application requires high-speed execution to beat the memory resume interrupt of the computer to which it is connected.

6. ADVANTAGES OF THE HARDWARE DESIGN

This section summarizes the advantages of the hardware design presented in the previous section.

6.1 REDUCED CONTEXT SWITCHING

One reason the Hardware Executive obtains high-speed execution of executive functions is the elimination of most context switching between the application and the executive and between the executive and itself. Context switching occurs when control of the computer resources is transferred from one process to another. It consists of writing the content of the registers associated with the former process into memory, and then loading the registers with new content read from memory appropriate for the next process. Slow execution results from the large number of memory accesses. The Hardware Executive uses at most only one processor register, the register used for executive

[15] Weinberger, Arnold, The Hybrid Associative Memory Concept, Computer Design, Jan 1971, p 77-85.

function operand transfer in machines not supporting memory-to-memory instructions. All its algorithms are performed internally. Thus there is no need to save more than one processor register when an executive function is invoked.

6.2 HIGH-SPEED EXECUTIVE TABLE SEARCHING

Another reason the Hardware Executive obtains high-speed execution is because it uses associative memory. Conventional executive implementations perform table searches sequentially, examining each table entry one at a time until a match is found. The internal associative memory permits all table entries to be examined in parallel.

6.3 EXECUTION SPEED INDEPENDENT OF LOAD

The Hardware Executive performs internal table searching by examining a single bit position of all table entries simultaneously. Thus the search time is the same regardless of the number of active table entries. Performance of the executive functions is easily predicted since it is not affected by load. This simplifies the task of the system designer.

6.4 INDEPENDENCE FROM HOST PROCESSOR INSTRUCTION SET

The Hardware Executive is connected to the host processor in the same way that a memory is connected. Functions are chosen by the selection of the appropriate memory address within the Hardware Executive address space. There is no requirement to augment or modify the host processor instruction set. This permits simple retrofit of the Hardware Executive to existing computers. It also permits the same basic unit to be attached to a large variety of different processor designs with only the memory interface logic changing from design to design.

7. COST VERSUS BENEFITS

This section considers the cost and benefits of the Hardware Executive in relation to other alternatives.

7.1 IDLE HARDWARE OBJECTION

An NRL report by Shore published in 1971 [16] presents an important issue concerning the application of associative processing as opposed to other types of processing. The report first states that "one criterion ... useful in comparing alternate designs is the degree to which the processing hardware is usefully busy." Terming the degree to which the hardware is usefully busy its "duty cycle", the report holds that associative processing should be employed only where this duty cycle is high. It further holds that associative processing is rarely justified since, with appropriate software techniques, a better duty cycle can be obtained by using a conventional computer design.

[16] Naval Research Laboratory Report 7364, Second Thoughts on Parallel Processing, by John E. Shore, 30 Dec 1971.

From the standpoint of the Hardware Executive by itself, its duty cycle is very low. Since it is used only when requested by the host, it stands idle most of the time. The duty cycle of the Hardware Executive actually decreases as its execution speed increases. But the Hardware Executive is not intended to be a stand-alone device. Like a high-speed multiplier, which in a conventional computer may also stand idle a considerable amount of time, the Hardware Executive is intended to increase significantly the overall performance of the entire computer system through a proportionally small increase in the total system hardware. The class of machines described in the report by Shore is totally associative. The Hardware Executive, on the other hand, is a hardware attachment to a conventional computer system.

7.2 RELATIVE SIZE OF THE HARDWARE

A 64-element Hardware Executive of the design presented requires approximately 310 to 330 integrated circuits, depending on the complexity of the host processor interface. For comparison, this is roughly halfway between an AN/AYK-14 CPU, with 424, and an AN/AYK-14 IOP, with 206. It should be noted, however, that the AN/AYK-14 processors employ many LSI components while the Hardware Executive is constructed entirely of SSI and MSI components. In general, LSI components are both mechanically larger and require more power. The typical power consumption of the Hardware Executive is 32.5 watts. This is well below the 89-watt requirement of the AN/AYK-14 CPU and the 44-watt requirement of the AN/AYK-14 IOP.

To gain some perspective, the chart below illustrates the relative costs of the various components of AN/AYK-14 computer configurations. The chart in figure 11 is based on 1979 prices for the AN/AYK-14.

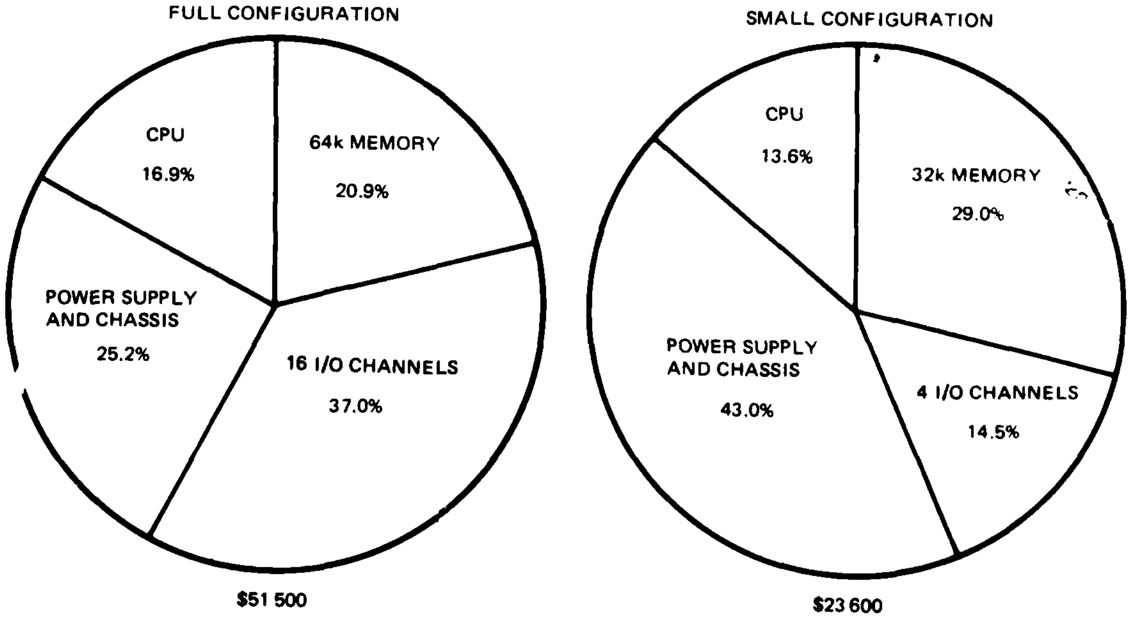


Figure 11. AN/AYK-14 price elements.

Seen in this perspective, the cost of the Hardware Executive is small in comparison with the total cost of the computer.

8. CONCLUSIONS

Large multiprocessing and distributed processing systems suffer from diminishing returns in system performance as additional processors are added. The slow execution speed of executive software is one of the principal causes of this phenomenon. This report describes the design of special-purpose hardware to overcome the software bottlenecks in executives. The design employs an associative memory for high-speed manipulation of executive tables. It minimizes the number of context switches by performing the executive functions in hardware external to the host processor. Context switching is also minimized by designing executive functions that are insensitive to pre-emption. The interface between the Hardware Executive and the host processor is achieved through the memory bus. This simple "memory-mapped" arrangement permits interfacing to a large number of existing computers without modification of their hardware or firmware. The interface also includes an interrupt for time-critical events when this feature is desired.

The logic design and fabrication of a Hardware Executive has been completed. At the time of this writing, the hardware is in the process of being interfaced to an AN/UYK-20 computer for testing.

REFERENCES

1. Brown, George E., et al, Operating System Enhancement through Firmware, 10th Annual Workshop on Microprogramming (Micro-10), 1977, p 119-133.
2. Burkhardt, Walter H., and Helmut E. Maier, Micos: A Microprogrammed Hierarchical Operating System Nucleus and Its Performance Comparison, 11th Annual Microprogramming Workshop (Micro-11), 1978, p 33.
3. Burkhardt, Walter H., and Ronald C. Randel, Design of Operating Systems with Micro-Programmed Implementation, Pittsburgh University, Sept 1973.
4. Chattergy, R., Microprogrammed Implementation of a Scheduler, 9th Annual Workshop on Microprogramming (Micro-9), 1976, p 15-19.
5. Fogarty, J.R., et al, Hardware Command and Control System Study: Final Report, Hughes Aircraft Co, Ground Systems Group, Fullerton CA, 18 Feb 1974, Navy contract N00123-73-C-2130 CDRL A002 submitted to Naval Electronics Laboratory Center.
6. Smith, William B., et al, SYMBOL - A large experimental system exploring major hardware replacement of software, SPRINT Joint Computer Conference, 1971, p 601-616.
7. Dearnley, P.A., Application level microcode to speed data base management, Computer Journal, British Computer Society, v 22, no 3, Aug 1979, p 200-202.
8. Naval Electronic Systems Command, Computer Program Performance Specification for Standard Executive for use with AN/UYK-20 and AN/AYK-14 Computers, SDEX/M; NAVSEA 0967-LP-598-2710, change 1, Dec 1980.
9. Chen, Tien Chi, Parallelism, Pipelining, and Computer Efficiency, Computer Design, Jan 1971, p 69-74.
10. Wulfinghoff, Donald R., Code Activated Switching: A Solution to Multiprocessing Problems, Computer Design, Apr 1971, p 67-71.
11. Mundell, J.L., et al, Conceptual Design of a Distributed Combat Direction System for a Modular Frigate (SEAMOD FFGX), prepared by System Exploration, Inc, under Navy Contract N00123-76-C-0787, Feb 1978, for Naval Ocean System Center, NOSC Technical Note 356.
12. Sperry Univac, Defense Systems Division, Navy Standard Computer Adaptation Study, v 1: Application Requirements, Navy Contract N66001-78-C-0258, July 1979, Final Report, prepared for NOSC.
13. Kuhns, Richard C., A Serial Data Bus System for Local Processing Networks, 18th IEEE Computer Society International Conference, Spring 1979, IEEE Catalog no 79CH1393-8C, p 266-271. (SHINPADS is also discussed in the Naval Engineering Journal, April and June 1979.)
14. NAVELEX User's Handbook: AN/UYK-20(V) Computer Support Software, NAVELEX 0967-LP-598-2020, v 2, change 6, Oct 1978.
15. Weinberger, Arnold, The Hybrid Associative Memory Concept, Computer Design, Jan 1971, p 77-85.
16. Naval Research Laboratory Report 7364, Second Thoughts on Parallel Processing, by John E. Shore, 30 Dec 1971.

BIBLIOGRAPHY

1. Ramamoorthy, C. V., and Benjamin W. Wah, A Design of a Fast Cellular Associative Memory for Ordered Retrieval, IEEE Transactions on Computers, v C-27, no 9, Sept 1978, p 800-815. (Additional comments from Ya I. Fat, ibid, v C-29, no 8, Aug 1980, p 756-757.)
2. Ruschitzka, Manfred, and R.S. Fabry, A Unifying Approach to Scheduling, Communications of the ACM, July 1977, v 20, no 7, p 469-477.
3. Stonebraker, Michael, Operating System Support for Database Management, Communications of the ACM, July 1981, v 24, no 7, p 412-418.
4. Vuillemin, Jean, A Data Structure for Manipulating Priority Queues, Communications of the ACM, Apr 1978, v 21, no 4, p 309-315.

APPENDIX A

EXECUTIVE FUNCTIONS

Executive function commands are executed by reading or writing dedicated memory address locations recognized by the Hardware Executive.

INITIALIZE

Purpose: To initialize the Hardware Executive.

Write: Anything since not used.

Function: Set all internal table locations to empty. All task and event registrations are lost.

RESERVE TASK

Purpose: To get an empty task table location for a new task.

Read: If positive, the index to an empty task table location. If negative, no more task table locations are available.

Function: Find the element with the lowest element address whose empty task bit is zero.

SUSPEND

Purpose: To exclude a task from dispatching.

Write: Task identifier.

Function: If the task was in the Ready or Running task state, it is now moved to the Suspended task state. If the task was in the Waiting task state, it is now moved to the Suspended-Waiting task state.

RESUME

Purpose: To discontinue task suspension.

Write: Task identifier.

Function: If the task was in the Suspended task state, it is now moved to the Ready task state. If the task was in the Suspended-Waiting task state, it is now moved to the Waiting task state. In the latter case, a check is made to determine if the task is the only task waiting on the semaphore. If so, the task is moved to the Ready task state and the Wait Semaphore field is cleared.

DISPATCH CHECK

Purpose: To determine the next task to run.

Read: If positive, the Task Identifier of the highest priority task in the Ready or Running task state. If negative, no task is in the Ready or Running task state.

Function: Find the highest priority task in the Ready or Running task state.

SIGNAL

Purpose: To announce that a resource is available.

Write: Semaphore identifier.

Function: Using the task semaphore and task priority fields, find the highest priority task waiting on the specified semaphore. Move that task from the Waiting task state to the Ready task state and clear the Wait Semaphore field. The signal command has no net effect if no tasks are waiting on the specified semaphore.

WAIT

Purpose: To place the running task in the wait state if a resource is not available.

Write: Semaphore identifier.

Function: Using the task semaphore fields, find any nonsuspended tasks waiting on the specified semaphore. If one or more are found, move the task invoking the WAIT function from the Running task state to the Waiting task state. Write the specified Semaphore Identifier into the Wait Semaphore field.

RESERVE NORMAL EVENT

Purpose: To get an empty normal event table location for a new normal event registration.

Read: If positive, the index to an empty normal event table location. If negative, no more normal event table locations are available.

Function: Find the normal event element with the lowest element address whose empty element bit is zero.

CAUSE NORMAL EVENT

Purpose: To associate a normal event with a normal event registration.

Write: Normal Event Identifier.

Function: Using the event level, class, subclass and code fields, associate the specified normal event with a unique event registration. Move the event registration from the Armed event state to the Triggered event state.

NORMAL EVENT SIGNAL CHECK

Purpose: To signal the semaphore of the highest-priority triggered normal event.

Read: If positive, the index into the normal event table of the highest-priority event registration in the Triggered event state. If negative, no normal event is in the Triggered event state.

Function: Attempt to signal the semaphore associated with the highest priority normal event registration in the Triggered event state. If successful, return the event to the Armed event state.

RESERVE TIME EVENT

Purpose: To get an empty time event table location for a new time event registration.

Read: If positive, the index to an empty time event table location. If negative, no more time event table locations are available.

Function: Find the element with the lowest-element address whose empty element bit is zero.

TIME EVENT SIGNAL CHECK

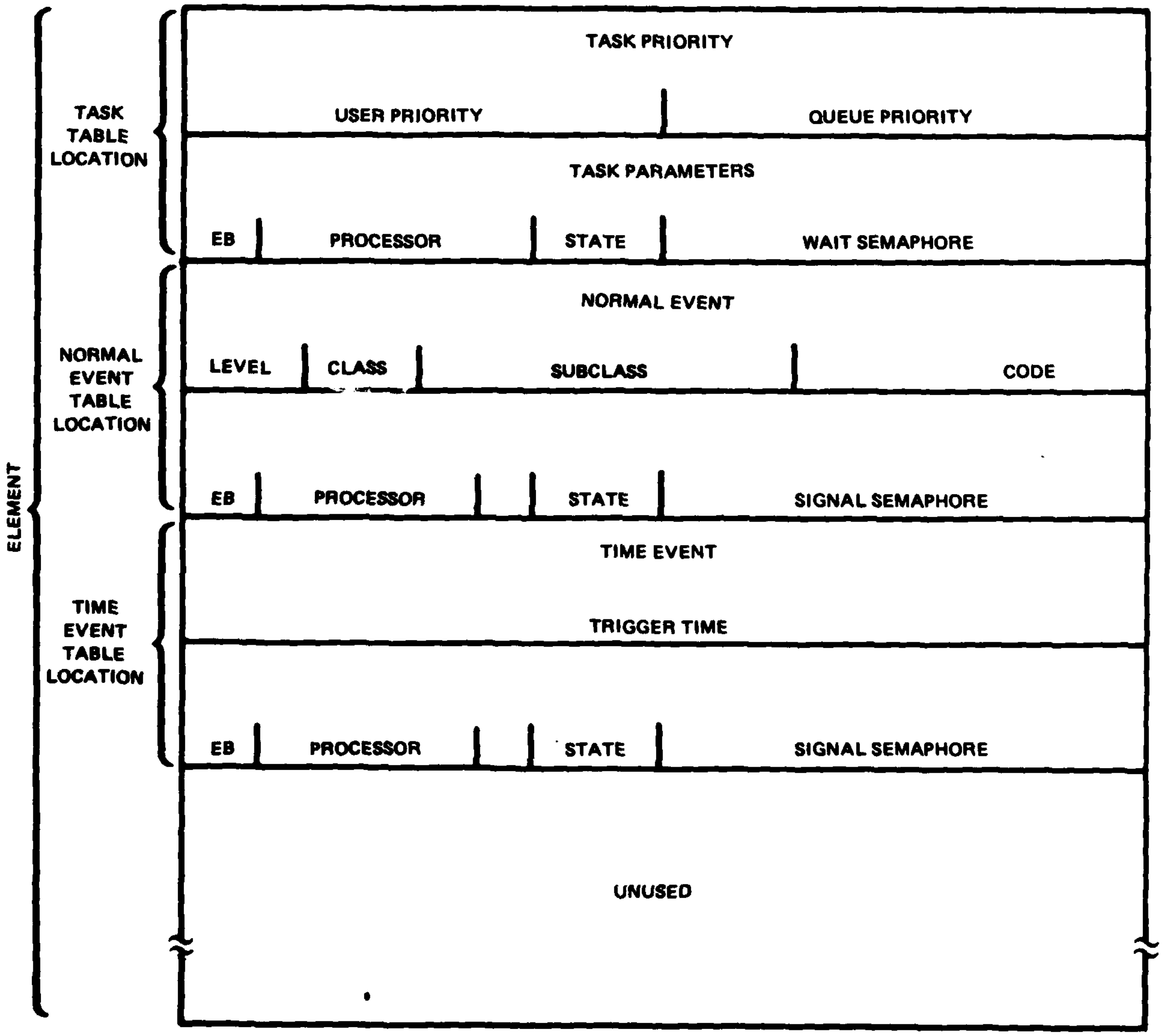
Purpose: To signal the semaphore associated with the highest-priority time event registration.

Read: If positive, the index into the time event table of the highest priority time event registration in the Triggered event state. If negative, there are no time event registrations in the Triggered event state.

Function: Attempt to signal the highest-priority time event in the Triggered event state. If successful, return the event to the Armed event state.

APPENDIX B

ELEMENT FORMAT



EB = EMPTY BIT

APPENDIX C

MICROINSTRUCTION FORMAT

Bit 31

Element Memory Write Control

0 Read

Reading is enabled for only those elements for which both the Element Capture Flip-Flop and the Bank Capture Flip-Flop are not set. The Element Memory Output for disabled elements during a read operation is one.

1 Write

Writing is enabled for only those elements for which both the Element Capture Flip-Flop and the Bank Capture Flip-Flop are not set. The Element Memory Output of all elements during a write operation is one.

Bits 30 - 28

Capture Flip-Flop Function

The results of these functions appear at the Capture Flip-Flop outputs at the beginning of the next cycle. Unless otherwise noted, the Element Memory Address Register is loaded with the contents of Bit Position field.

0 0 0 No Change

0 0 1 Clear

All Bank and Element Capture Flip-Flops are cleared.

The Clear function is used to initialize the Capture Flip-Flops before a search operation.

0 1 0 Set

All Bank Capture Flip-Flops are set. The Element Capture Flip-Flops are unchanged.

0 1 1 Prioritize

All Bank Capture Flip-Flops are set except the flip-flop for the bank containing the Element Capture Flip-Flop that has the lowest element address and is not set. All Element

Capture Flip-Flops of the selected bank are set except the flip-flop of the selected element. The Element Capture Flip-Flops in all banks are set identically. The Priority Register is loaded with the selected element address. It is cleared when no element is selected.

The Prioritize function normally follows a winnow function to select a unique element from all those elements not captured after a search operation.

1 0 0 Exclusive Winnow

If an Element Capture Flip-Flop is set, it remains set. Otherwise, it is set if the output of the respective serial arithmetic unit is one and in so doing does not leave all Element Capture Flip-Flops set. The Bank Capture Flip-Flops are cleared.

The Exclusive Winnow function is used when searching for a field with the lowest numeric value. Since at least one field satisfies the search, the function inhibits the setting of any Element Capture Flip-Flops if so doing will leave all of them set.

1 0 1 Inclusive Winnow

If an Element Capture Flip-Flop is set, it remains set. Otherwise, it is set if the output of the respective serial arithmetic unit is one. The Bank Capture Flip-Flops are cleared.

The Inclusive Winnow function is used when searching for a field whose contents are identical to some key. Since possibly no field satisfies the search, the function permits all the Capture Flip-Flops to be simultaneously set.

1 1 0 Exchange Register Addressed

All Bank Capture Flip-Flops are set except the flip-flop for the bank selected by bits 5 through 3 of the Exchange Register. All Element Capture Flip-Flops in each bank are set except the flip-flop selected by bits 2 through 0 of the Exchange Register. The Element Capture Flip-Flops in all banks are set identically.

The Exchange Register Addressed function permits the specification of the element address by the host processor Write Data, as required for the Suspend and Resume state change executive functions.

1 1 1 Host Processor Address Selected

All Bank Capture Flip-Flops are set except the flip-flop for the bank selected by bits 5 through 3 of the Exchange Register. All Element Capture Flip-Flops in each bank are set except the flip-flop selected by bits 2 through 0 of the Exchange Register. The Element Capture Flip-Flops in all banks are set identically. The Host Processor Address Selected function also places CPU Address Register bits 3 through 0 into Element Memory Address Register bits 7 through 4. Microinstruction bits 11 through 8 supply Element Memory Address Register bits 3 through 0 as usual.

The Host Processor Address Selected function selects a word within an element of the element memory for direct CPU access.

Bits 27 - 24

Register Function

0 0 0 0 No Change

0 0 0 1 Increment Time Counter

0 0 1 0 Increment Sub-Priority Counter

0 0 1 1 Decrement Sub-Priority Counter

0 1 0 0 Load Upper Time Counter from Exchange Register

0 1 0 1 Load Lower Time Counter from Exchange Register

0 1 1 0 Load Sub-Priority Counter from Exchange Register

0 1 1 1 Element Memory Loop

The Element Memory Loop function connects the data input of the memory associated with each element to the serial adder output generated on the preceding cycle for the respective element.

1 0 0 0 No Change

1 0 0 1 Shift Exchange Register Right

1 0 1 0 Store Sub-Priority into Exchange Register

1 0 1 1 Store Upper Time Counter into Exchange Register

1 1 0 0 Store Lower Time Counter into Exchange Register

1 1 0 1	Store Priority Register into Exchange Register
1 1 1 0	Store Data Input from Host Processor into Exchange Register
1 1 1 1	Store Address Input from Host Processor into Exchange Register

Bits 23 - 22

Serial Arithmetic Function

Initialization of the Output and Carry flip-flops of the serial arithmetic unit occurs immediately. The results of the Add and Subtract functions are not available until the beginning of the next cycle.

0 0	Clear Output, Clear Carry
0 1	Clear Output, Set Carry
1 0	Add
1 1	Subtract

Bits 21 -.19

Binary Operand Select

If Bit 23 is zero, the output is cleared regardless of the contents of Bits 21 through 19. The results of these functions are not available at the data inputs of the element memory and the serial arithmetic unit until the beginning of the next cycle.

0 0 0	Zero
0 0 1	Two
0 1 0	Sub-Priority

The 16 Sub-Priority Counter bit positions are addressed by the least-significant bits of the Bit Position field.

The Sub-Priority Counter supplies the contents of the task element subpriority fields when tasks are moved to the ready or running task state. It is used to implement queuing within user-supplied priority levels.

0 1 1 One

1 0 0 Time Plus Exchange Register

The bit positions of the Time Counter and the Exchange Register are addressed by the least-significant bits of the Bit Position field. They must be addressed sequentially from least to most significant for the proper sum to be computed serially.

> This function is used to add a time interval supplied by the user through the Exchange Register to the current time maintained in the Time Counter to form the trigger time for time events.

1 0 1 Time

The 32 Time Counter bit positions are addressed by the least-significant bits of the Bit Position field.

> The Time Counter provides a time base for time-dependent and time-critical events. The output of the Time Counter is compared with the contents of the trigger time registered with timed events to determine whether the event should be triggered.

1 1 0 Exchange Register

The 16 Exchange Register bit positions are addressed by the least-significant bits of the Bit Position field.

1 1 1 Zero

Reserved for the future.

Bits 18 - 16

Branch and Host Interface Control

0 0 0 Clear Host Interface Busy

Bit 0 of the next microinstruction address is zero. Host Interface Busy is cleared.

0 0 1 Pulse Host Interface Data Available

Bit 0 of the next microinstruction address is zero. Host Interface Data Available are strobed.

0 1 0 Clear Tick Flag

Bit 0 of the next microinstruction address is zero. The Tick Flag is cleared.

The Tick Flag is set by the tick interval counter. It is cleared by the microprogram after completing the triggering of timed events.

0 1 1 Pulse Host External Interrupt

The External Interrupt line is pulsed.

The External Interrupt line is pulsed by the microprogram when a time-critical event is identified.

1 0 0 Zero

Bit 0 of the next microinstruction address is zero.

1 0 1 One

Bit 0 of the next microinstruction address is one.

1 1 0 All Capture Flag

Bit 0 of the next microinstruction address is zero if all the Element Capture Flip-Flops are set. Otherwise, it is one.

1 1 1 Transform

Bit 0 of the next microinstruction address is one. Bits 8 through 1 of the next microinstruction address are the logical bitwise OR of the command transformed from the host processor memory address lines and bits 7 through 0 of the microinstruction.

Bits 15 - 8

Bit Position

The Bit Position field specifies the element bit position within the element memory by loading the Element Memory Address Register. Exceptions are noted in the description of the Capture Flip-Flop Function field. It also specifies the register bit position when the Binary Operand Select field selects a register. The specified bit position address is available to the element memory and selected register at the beginning of the next cycle.

Bits 7 - 0

Next Microinstruction Address

The Next Microinstruction Address field specifies bits 8 through 1 of the next microinstruction address. Bit 0 of the next microinstruction address is generated by the Branch and Bus Control field.